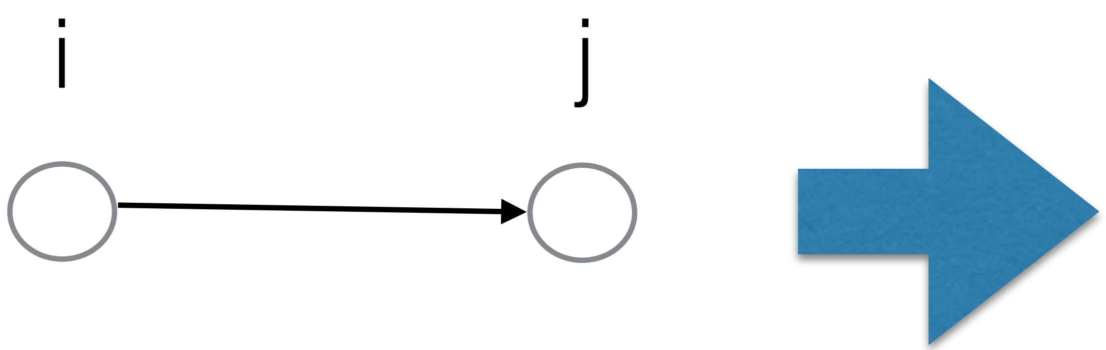
Strongly Connected Components, Dijkstra

Lecture19

Topological Sort

```
TOPOLOGICALSORT(G):
    add vertex s
    for all vertices v ≠ s
        add edge s→v
        status(v) ← NEW
    TOPOSORTDFS(s)
    for i ← 1 to V
        S[i] ← POP
    return S[1..V]
```

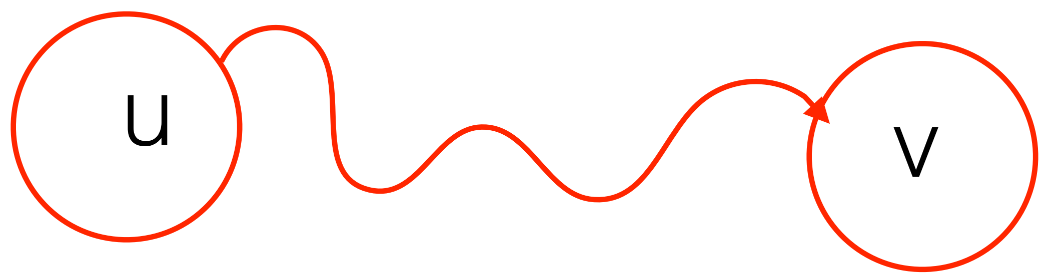
```
TOPOSORTDFS(v):
    status(v) ← ACTIVE
    for each edge v→w
        if status(w) = NEW
            PROCESSBACKWARDDFS(w)
        else if status(w) = ACTIVE
            fail gracefully
    status(v) ← DONE
    PUSH(v)
    return TRUE
```



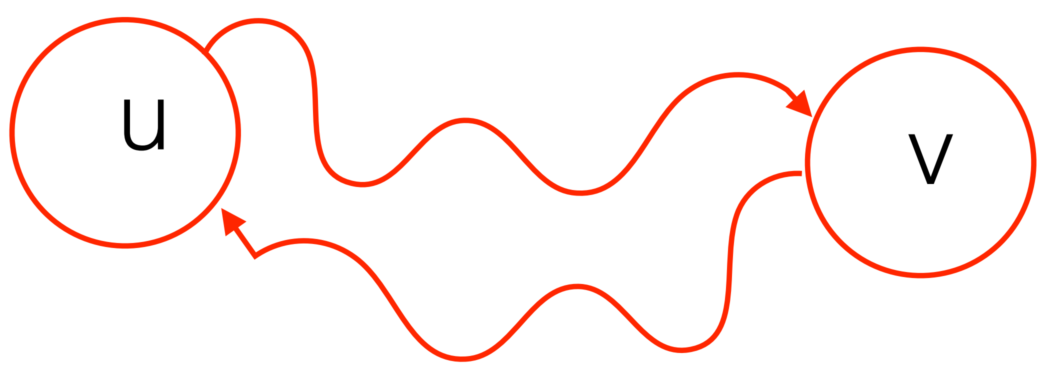
$i<j$

Strong Connectivity

In directed graph vertex u can reach vertex v iff
there is a directed path from u to b
reach(u) = set of vertices u can reach

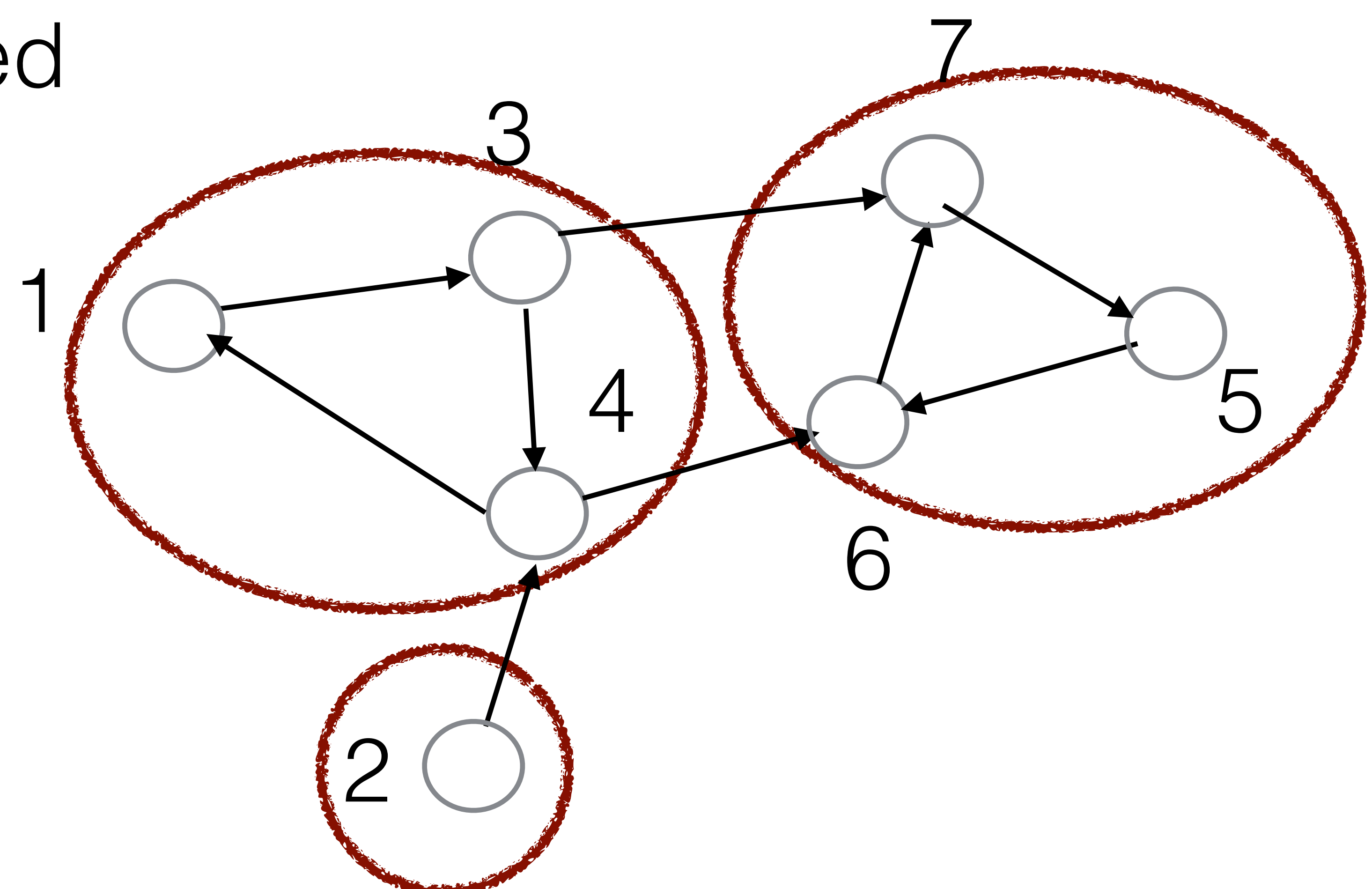


u and v are strongly connected if u can reach v and v can reach u



Strong Connectivity, SCC

- Strong connectivity is an equivalence relation
- Equivalence classes are called strongly connected components
- If G has a single strongly connected component: strongly connected



Strong Connectivity, SCC

- Strong connectivity is an equivalence relation
- Equivalence classes are called strongly connected components
- If G has a single strongly connected component: strongly connected
- When is G a DAG?
- No two vertices strongly connected
- Every SCC is a single vertex

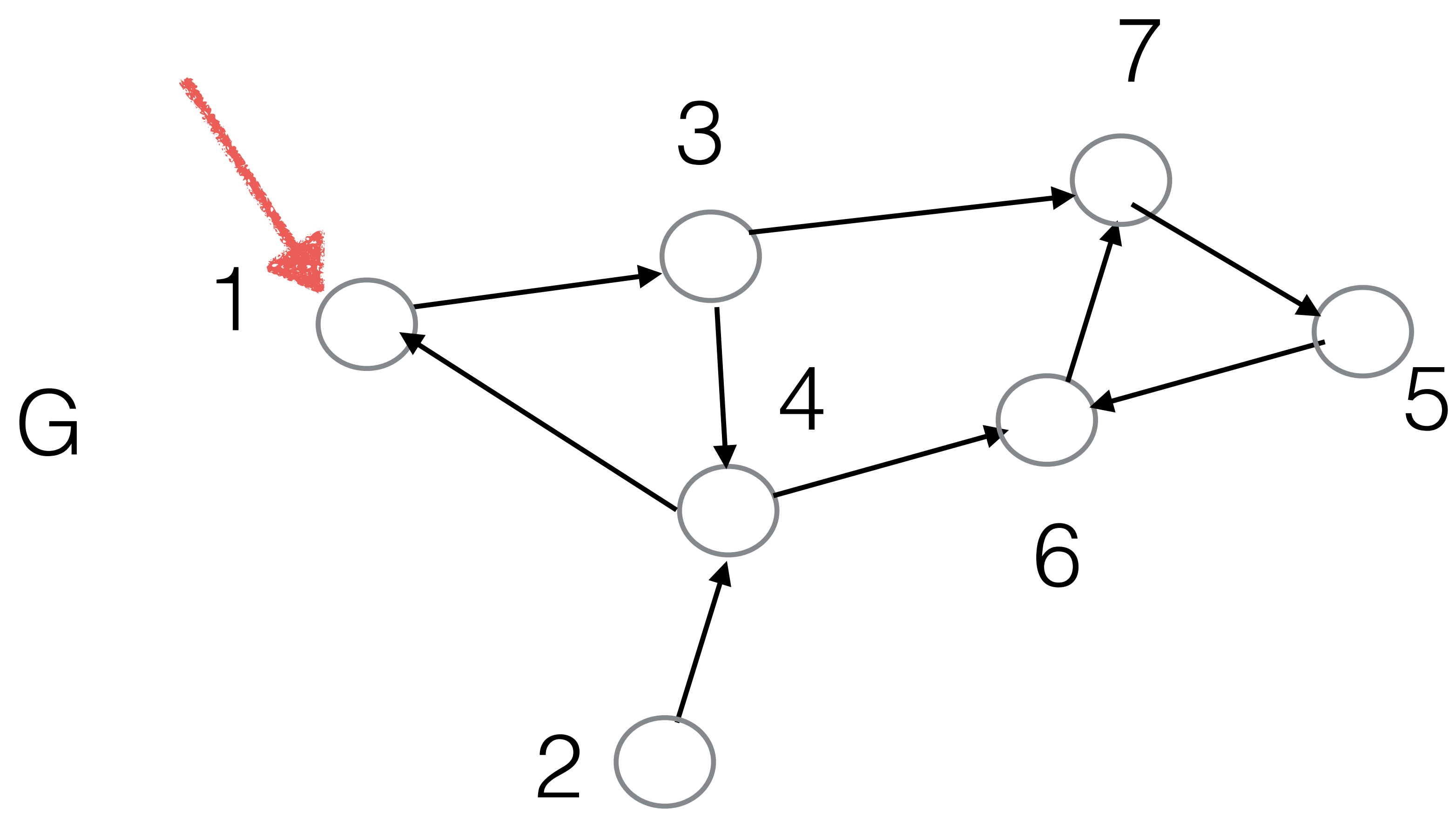
Strong Connectivity, SCC

- How to compute SCC of vertex u in O(|V|+|E|) time?

DFS(G,u) gives us Reach(u)

DFS(G^{rev},u) gives us all the stuff that can reach u

Take intersection of both for SCC



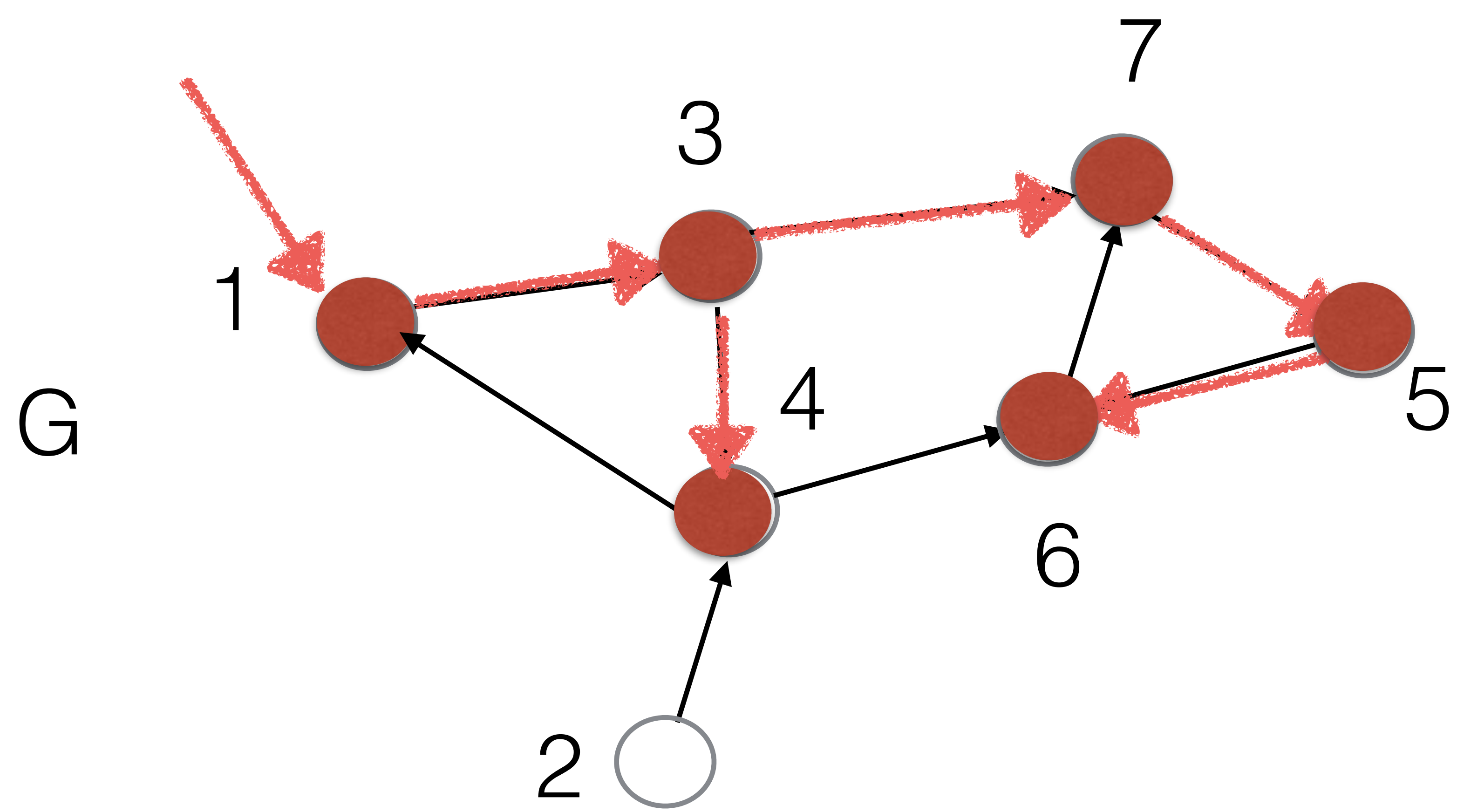
Strong Connectivity, SCC

- How to compute SCC of vertex u in O(|V|+|E|) time?

DFS(G,u) gives us Reach(u)

DFS(G^{rev},u) gives us all the stuff that can reach u

Take intersection of both for SCC



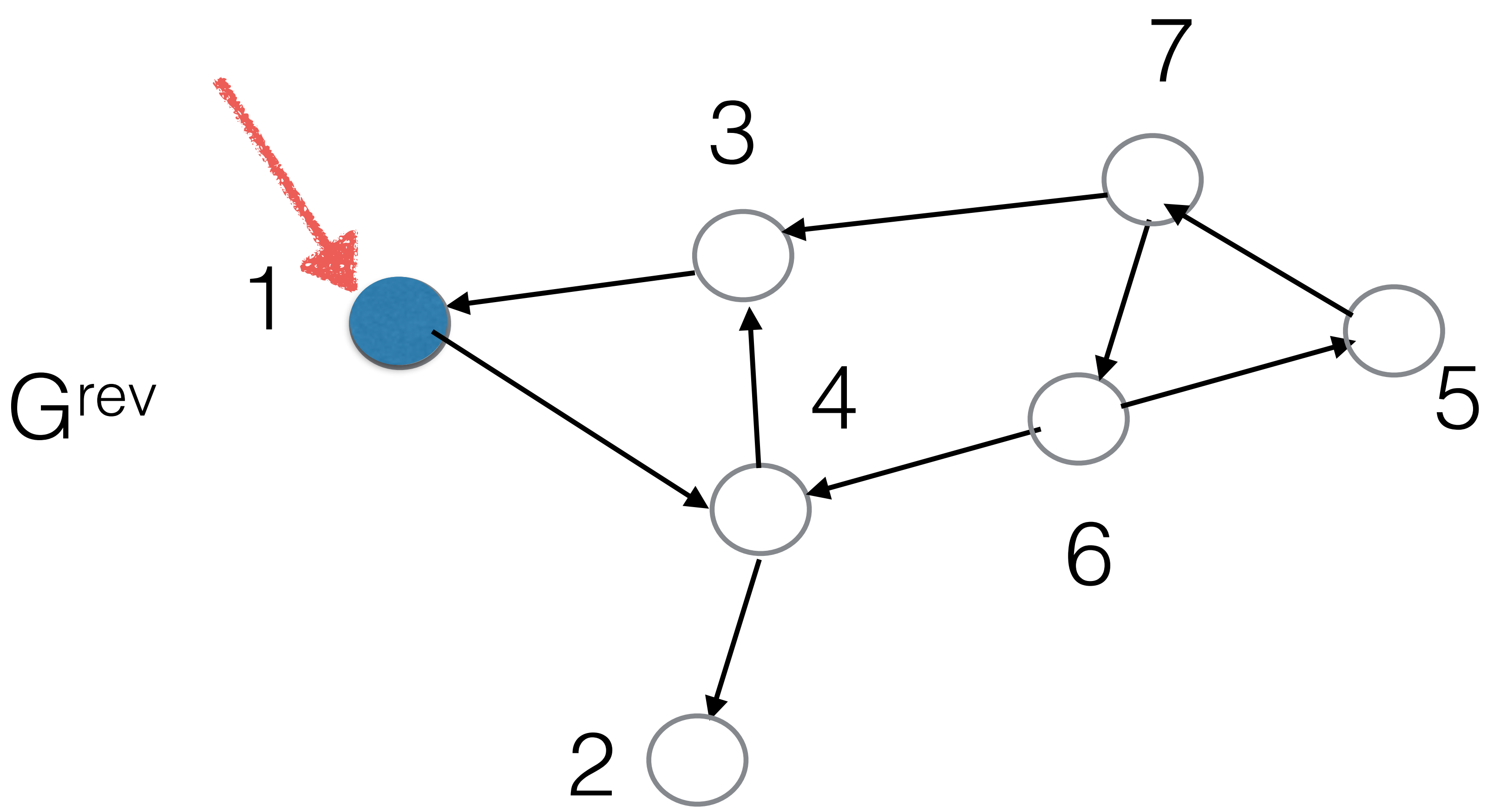
Strong Connectivity, SCC

- How to compute SCC of vertex u in O(|V|+|E|) time?

DFS(G,u) gives us Reach(u)

DFS(G^{rev},u) gives us all the stuff that can reach u

Take intersection of both for SCC



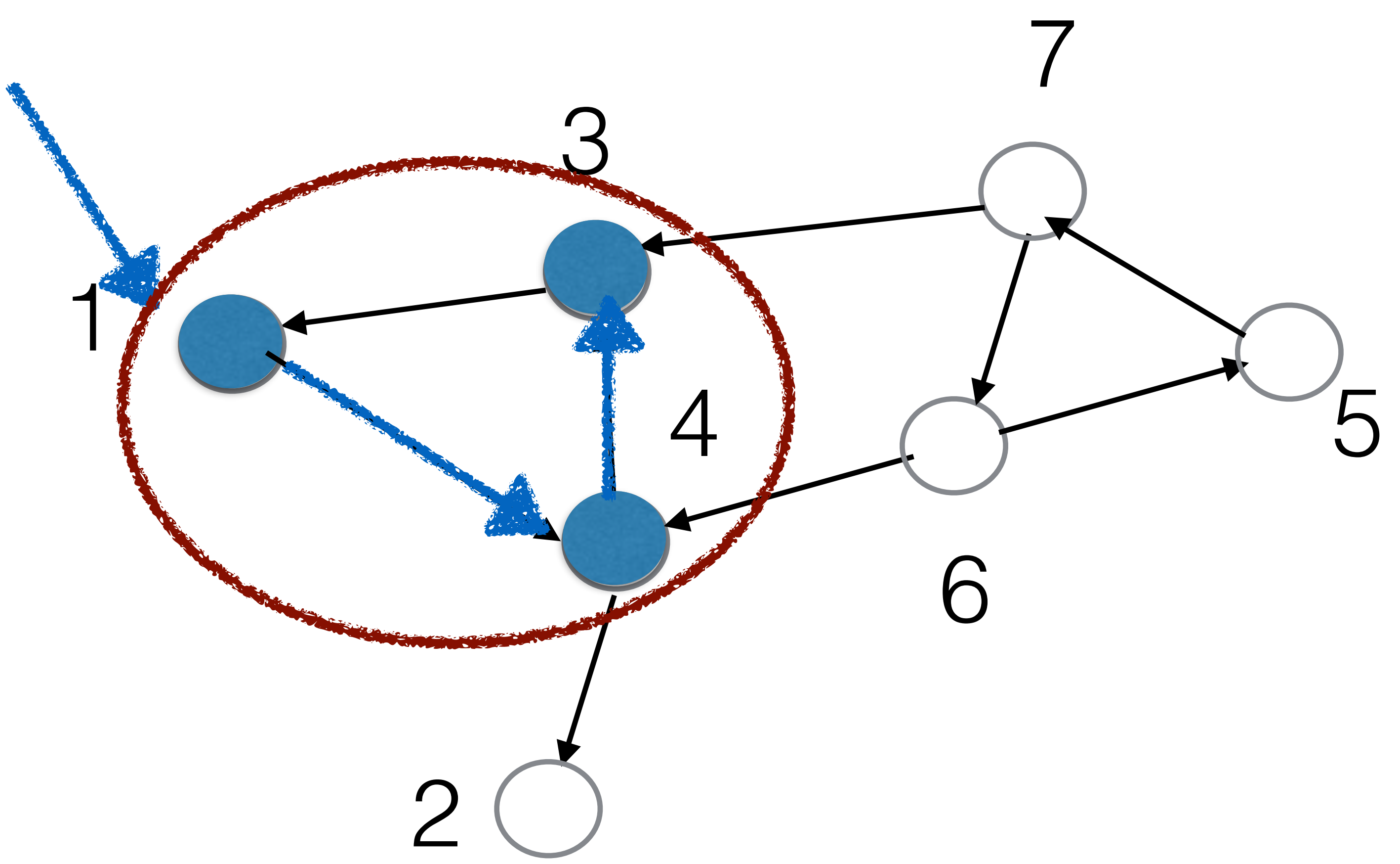
Strong Connectivity, SCC

- How to compute SCC of vertex u in O(|V|+|E|) time?

DFS(G,u) gives us Reach(u)

DFS(G^{rev},u) gives us all the stuff that can reach u

Take intersection of both for SCC

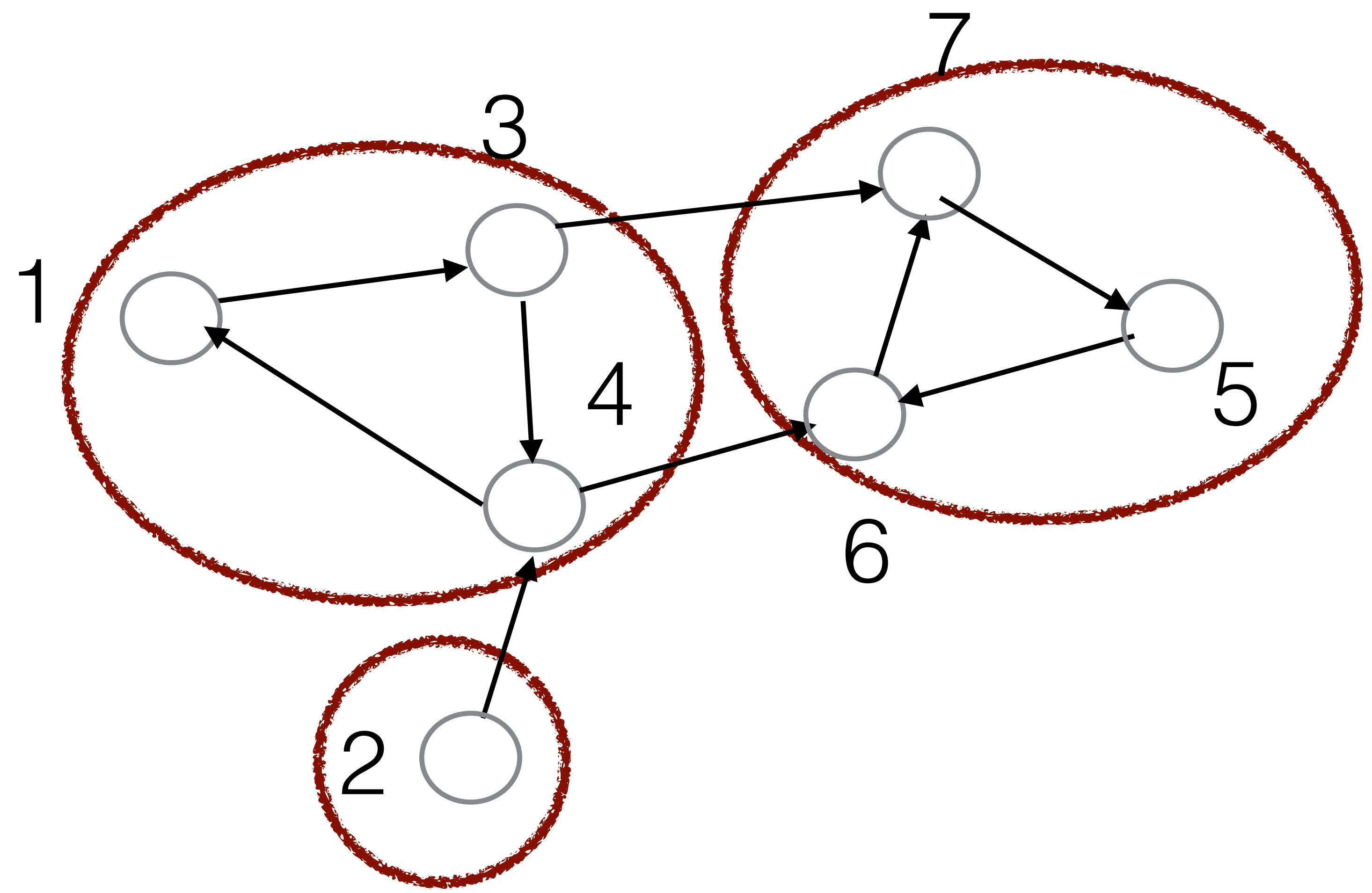


Strong Connectivity, SCC

- How to compute SCC of vertex u in $O(|V|+|E|)$ time?
- Compute Reach(u) with DFS on G in $O(|V|+|E|)$
- Compute $Reach^{-1}(u) = \{v: u \text{ is in } Reach(v)\}$ with DFS on reverse graph G^{rev} in $O(|V|+|E|)$
- SCC is the intersection of the two sets (mark vertices that have been visited on the first DFS).
- How to compute all SCC of a graph?
- Naive: $O(|V||E|)$ time (for every vertex compute its component).
- Can we do better?
- Combine all the DFS into one.

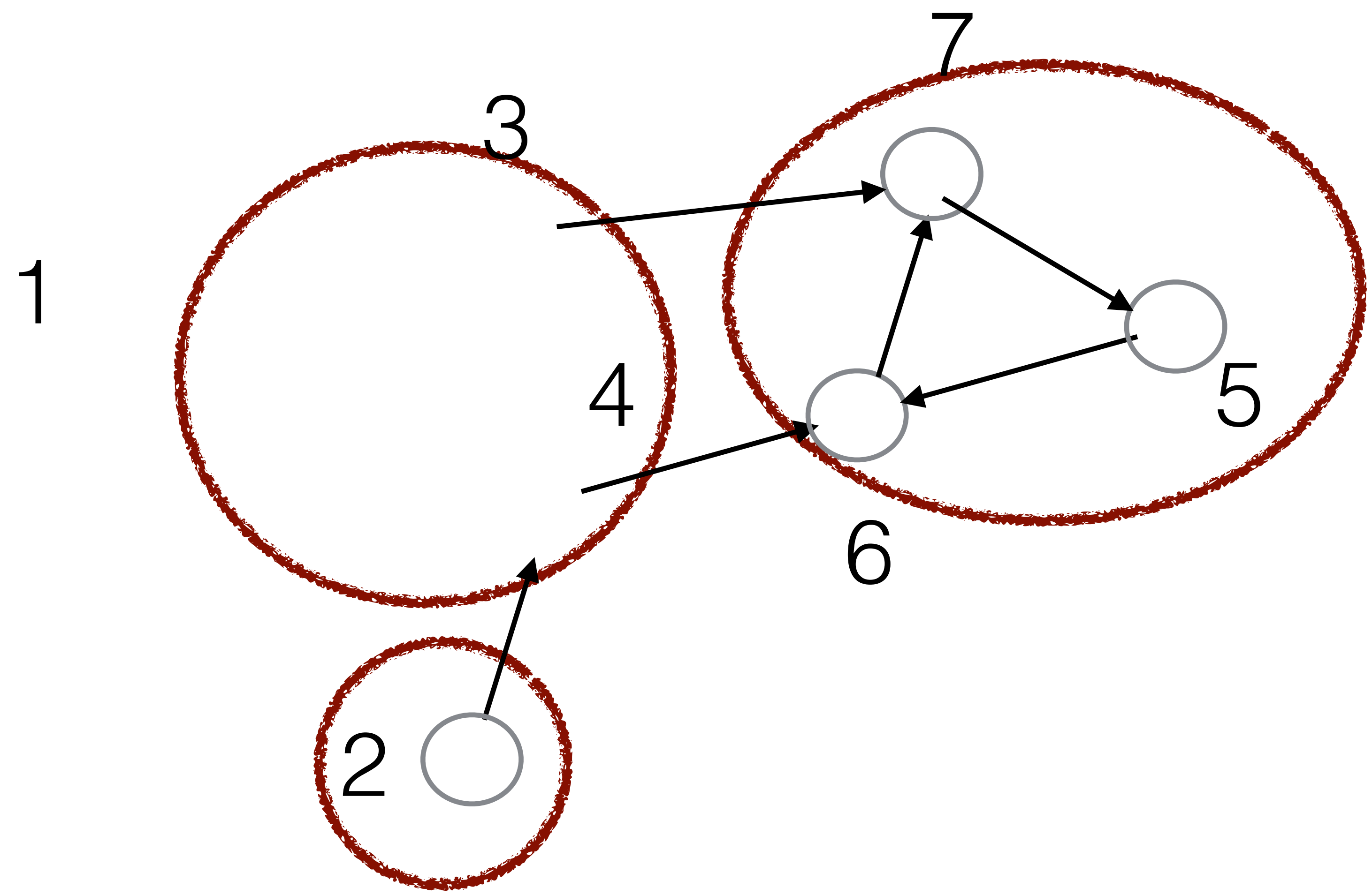
SCC Graph

For every directed graph G, scc(G) is another (meta)graph: Contract each SCC of G in one vertex and collapse parallel edges



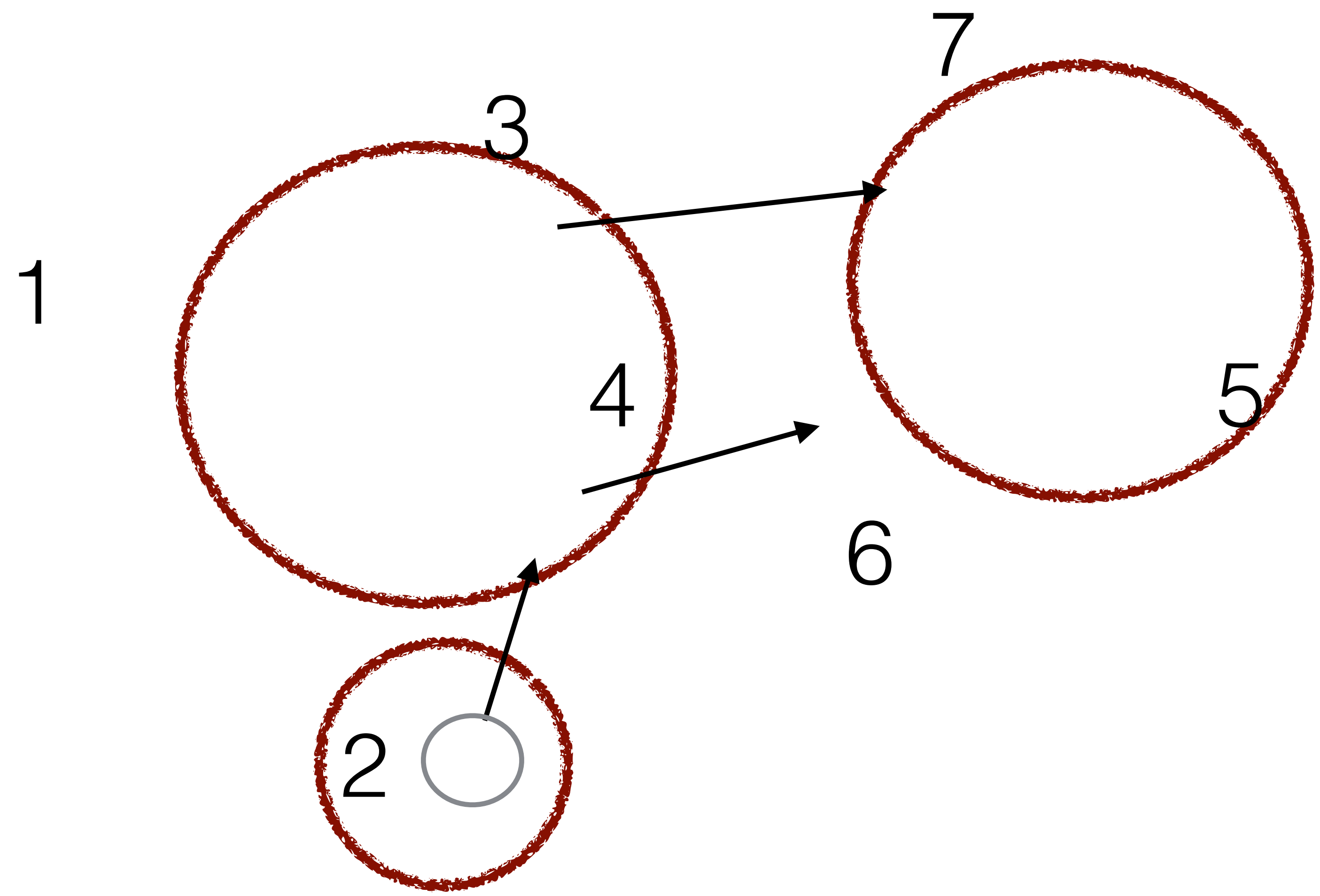
SCC Graph

For every directed graph G, scc(G) is another (meta)graph: Contract each SCC of G in one vertex and collapse parallel edges



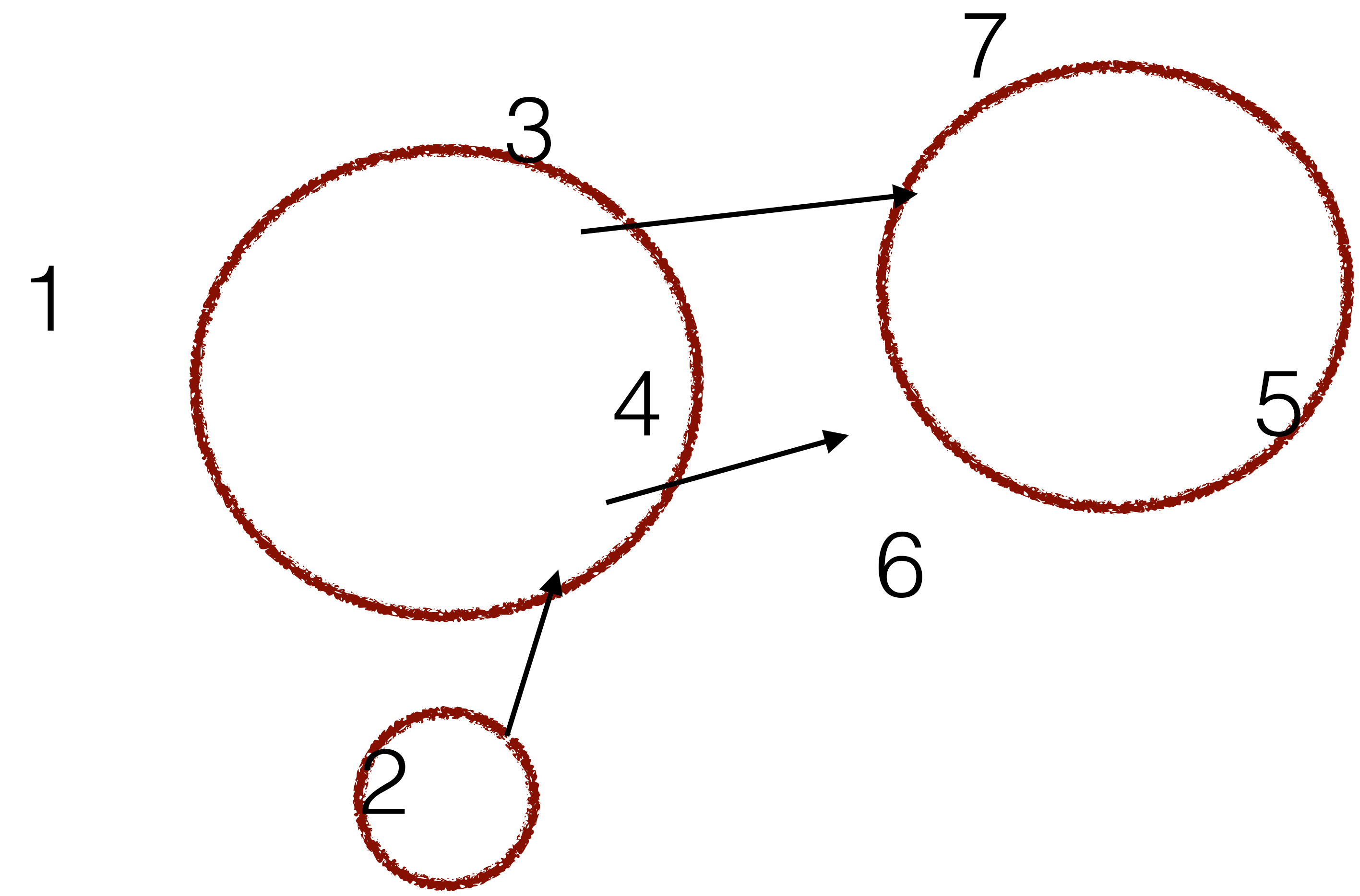
SCC Graph

For every directed graph G, scc(G) is another (meta)graph: Contract each SCC of G in one vertex and collapse parallel edges



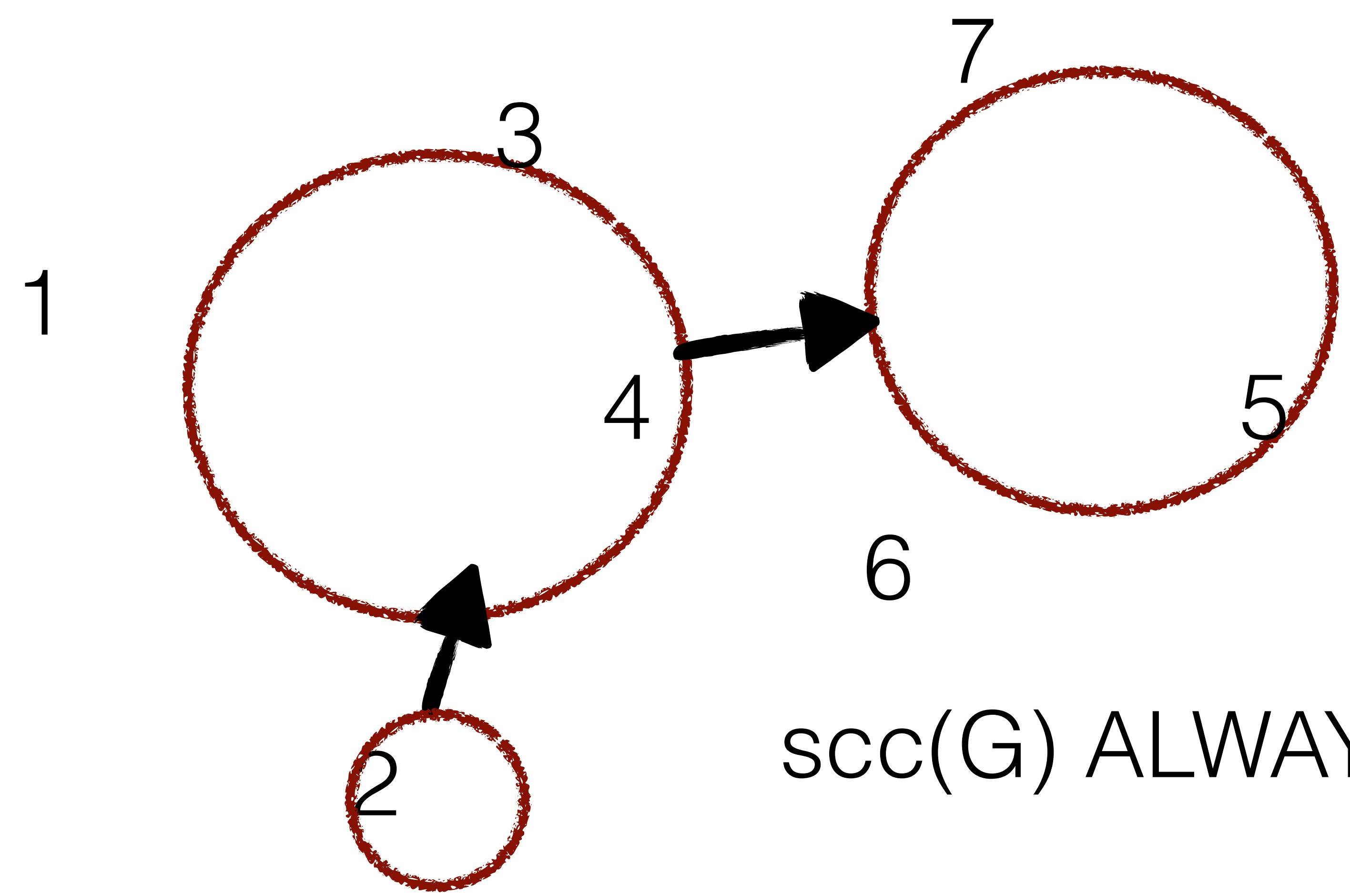
SCC Graph

For every directed graph G, scc(G) is another (meta)graph: Contract each SCC of G in one vertex and collapse parallel edges



SCC Graph

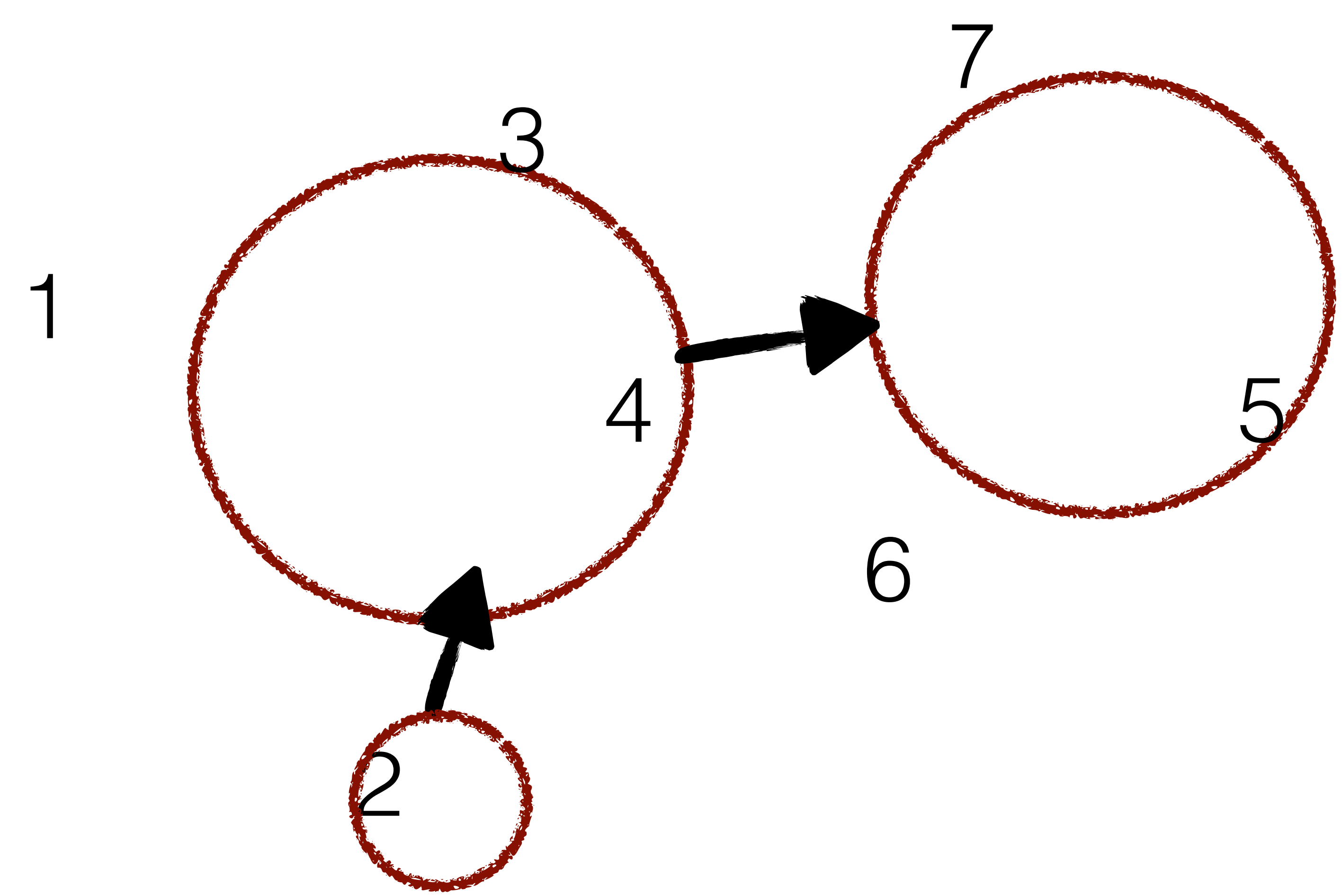
For every directed graph G, scc(G) is another (meta)graph: Contract each SCC of G in one vertex and collapse parallel edges



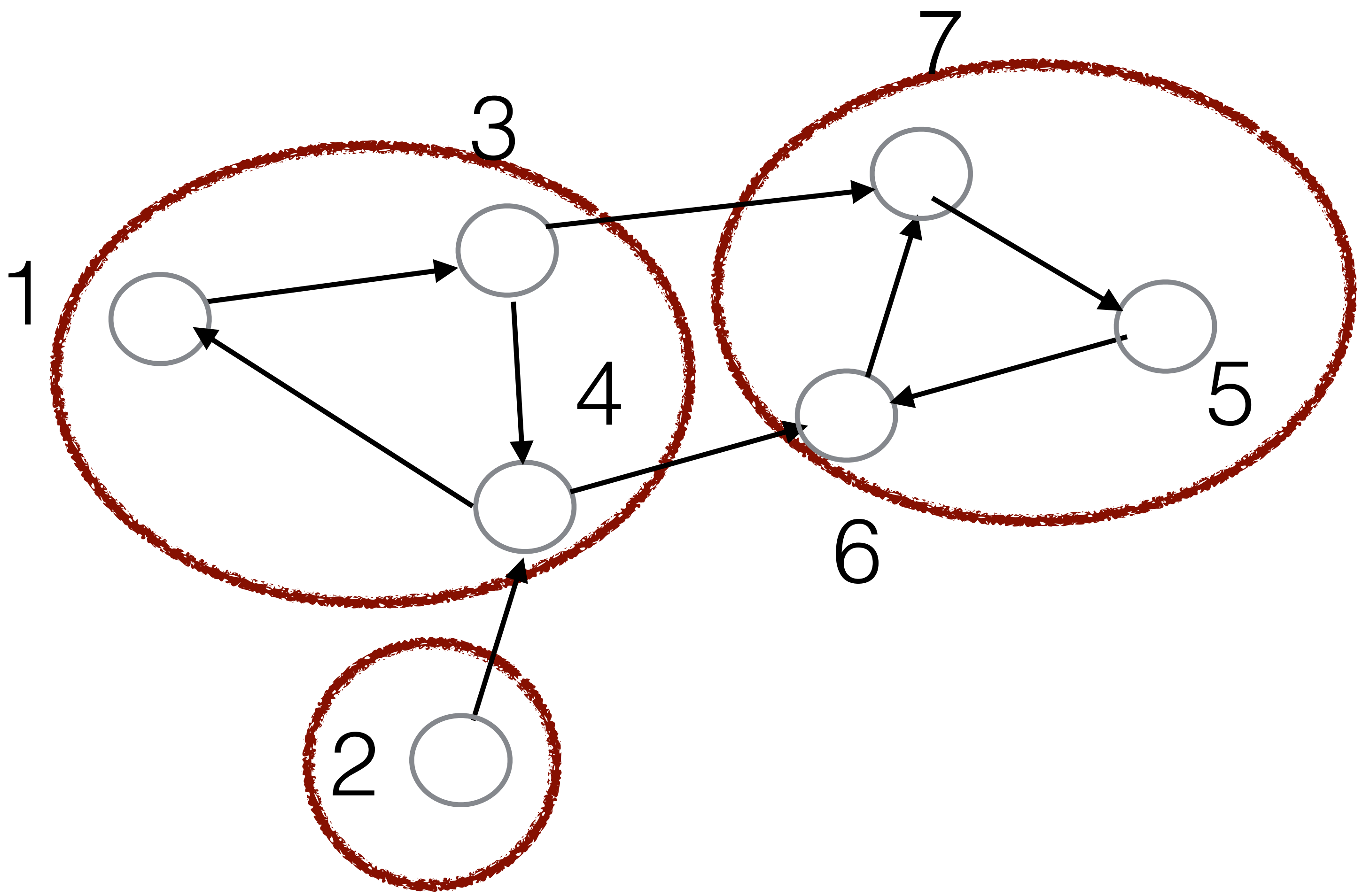
scc(G) ALWAYS A DAG!

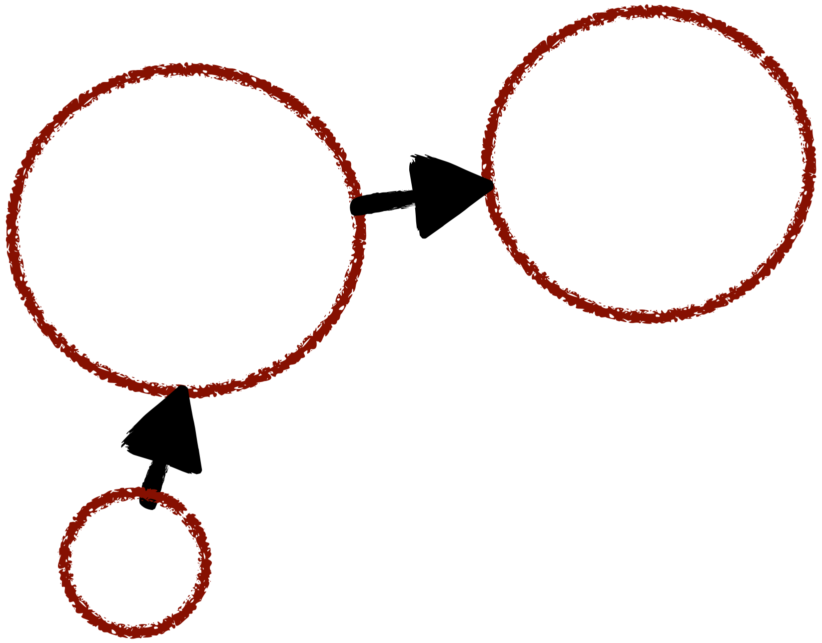
SCC Graph

We want to find all SCC, namely compute scc(G) graph in linear time



SCC Graph





- What if I try to do it recursively?
- Find a sink (or source) component of scc(G), remove it and recurse.

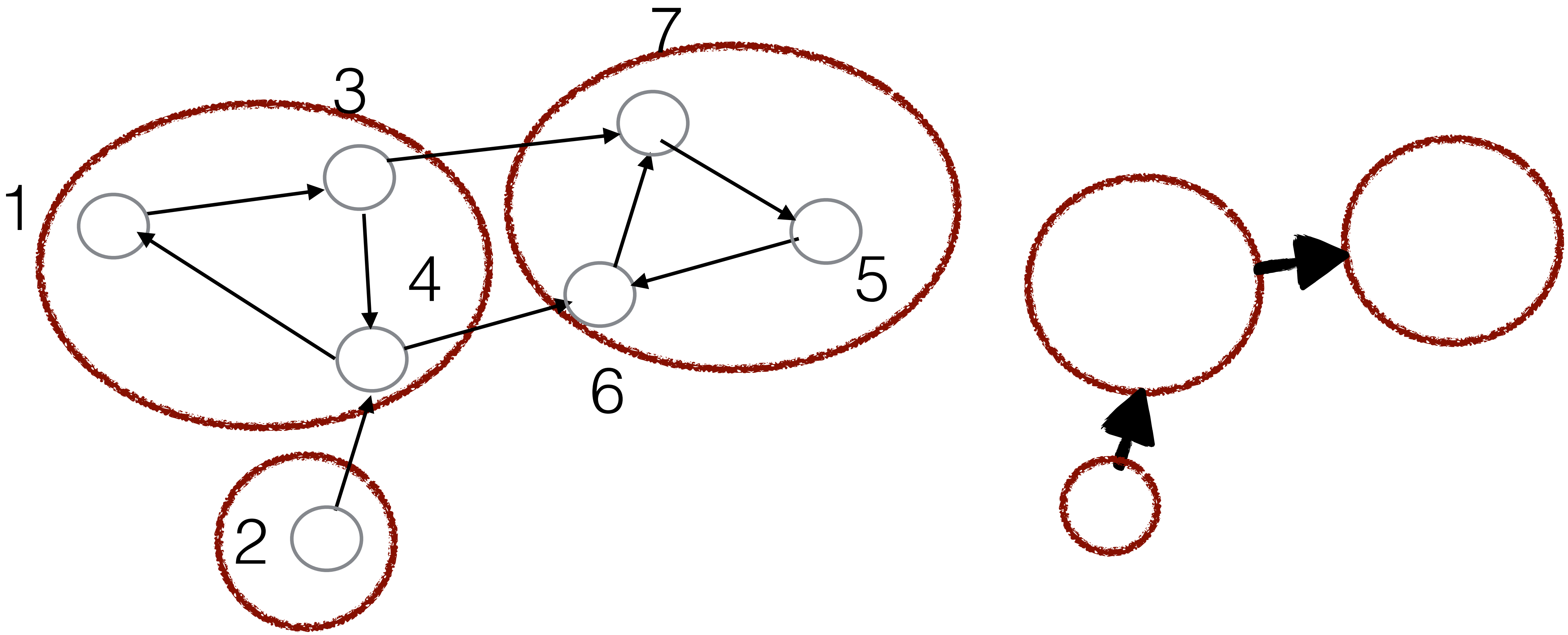
SCC Graph

Can compute all the SCC:

```
StrongComponents(G):
    count ← 0
    while G is non-empty
        count ← count + 1
        v ← any vertex in a sink component of G
        C ← OneComponent(v, count)
        remove C and incoming edges from G
```

How to find a vertex in a sink component?

SCC Graph



- What if I try to do it recursively?
- Find a sink (or source) component of scc(G), remove it and recurse.
- Last time for DAGS: first vertex DONE in DFS is a sink!

Finding SCC

- **Claim**: Last vertex DONE is in a source component of scc(G).

```
DFSAll(G):
    for all vertices v
        unmark v
    clock ← 0
    for all vertices v
        if v is unmarked
            clock ← DFS(v, clock)
```

```
DFS(v, clock):
    mark v
    for each edge v→w
        if w is unmarked
            clock ← DFS(w, clock)
    clock ← clock + 1
    finish(v) ← clock
    return clock
```

Running time?
Do something for every SCC, will give us something quadratic on worst case (e.g. DAG)
But the vertices are in the correct order!

Finding SCC

- **Claim**: For any edge v → w in G, if finish(v) < finish(w), then v and w are strongly connected in G.

Finding SCC

- SCC in O(|V|+|E| time) (just two DFS one in G and one in reverse!)

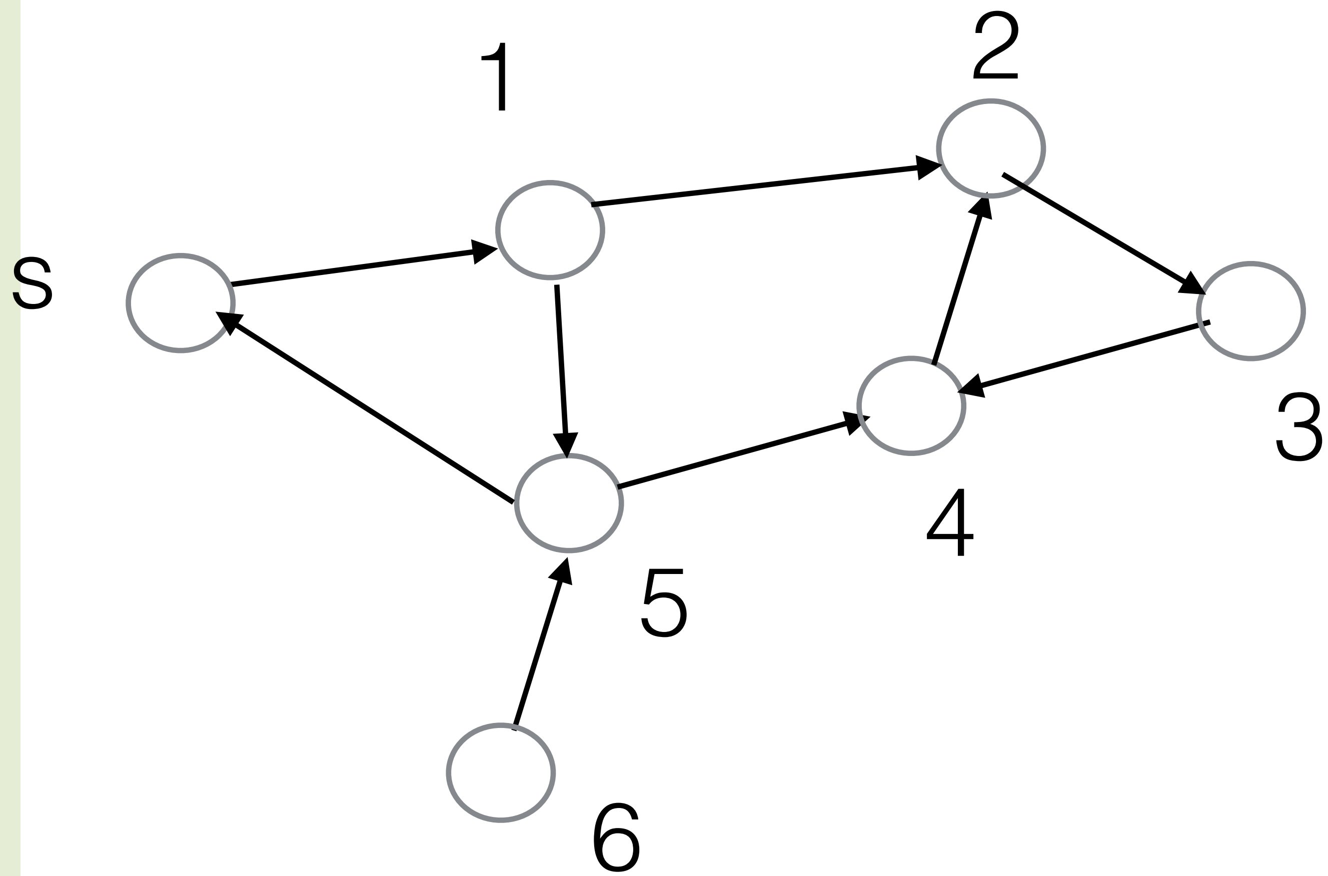
```
KosarajuSharir(G):
    ⟨⟨Phase 1: Push in finishing order⟩⟩
    unmark all vertices
    for all vertices v
        if v is unmarked
            clock ← RevPushDFS(v)
    ⟨⟨Phase 2: DFS in stack order⟩⟩
    unmark all vertices
    count ← 0
    while the stack is non-empty
        v ← Pop
        if v is unmarked
            count ← count + 1
            LabelOneDFS(v, count)
```

```
RevPushDFS(v):
    mark v
    for each edge v→u in rev(G)
        if u is unmarked
            RevPushDFS(u)
    Push(v)
```

```
LabelOneDFS(v, count):
    mark v
    label(v) ← count
    for each edge v→w in G
        if w is unmarked
            LabelOneDFS(w, count)
```

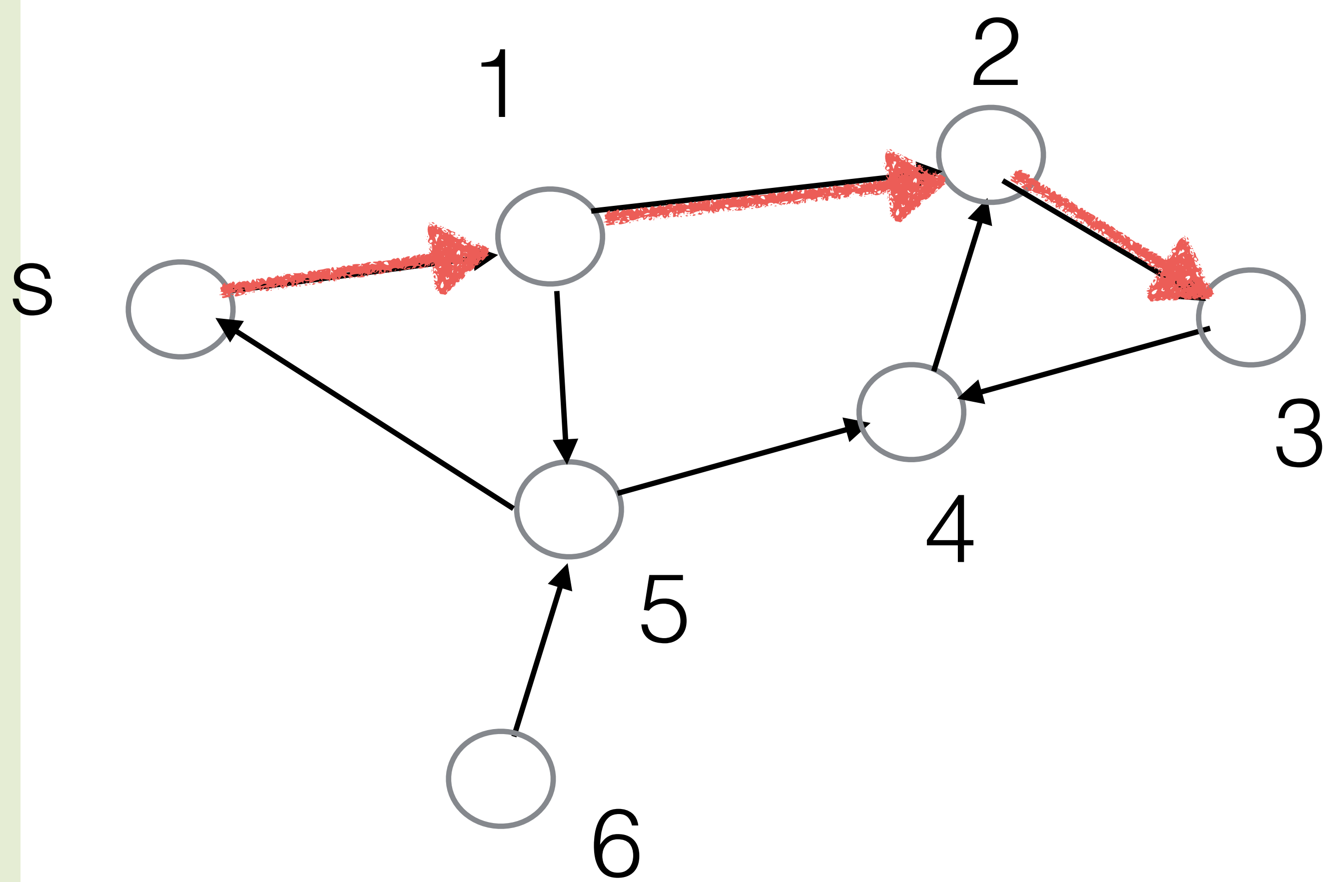
Single Source Shortest Paths

Shortest Paths

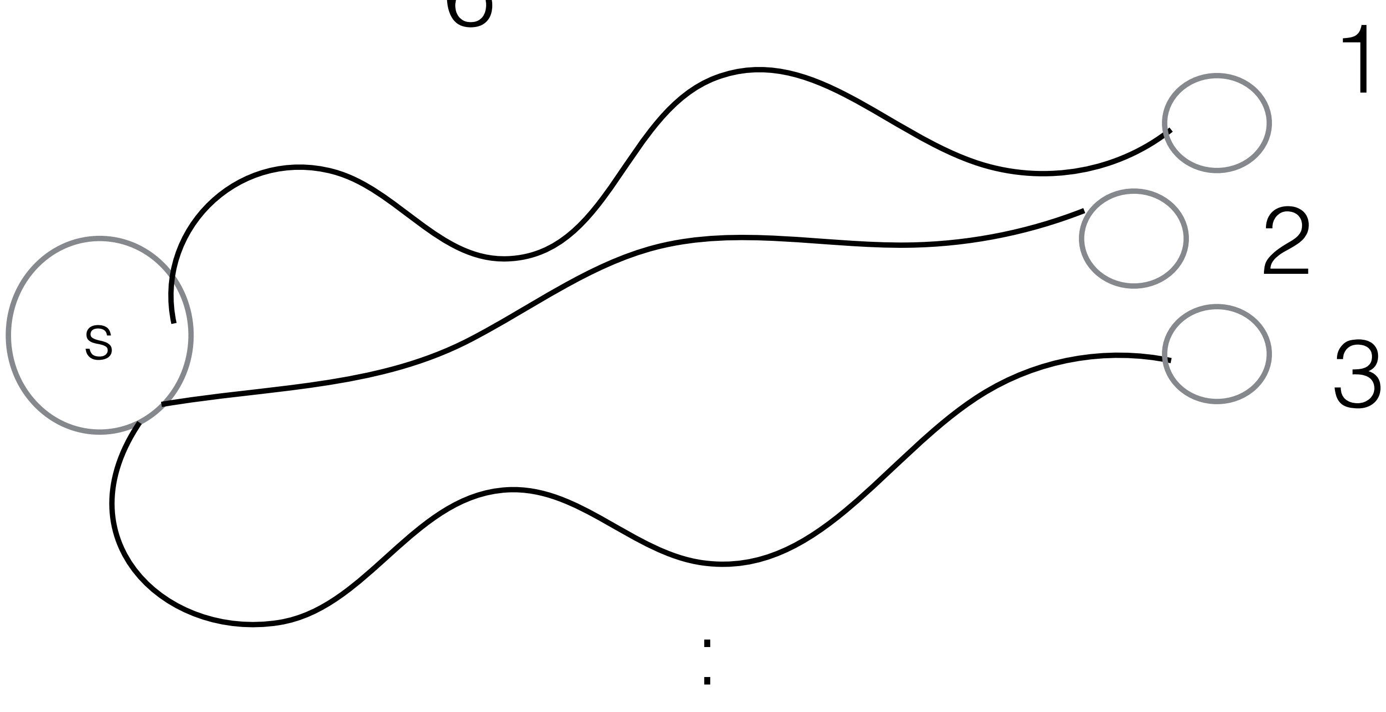


- Single source shortest path (one s, all t)

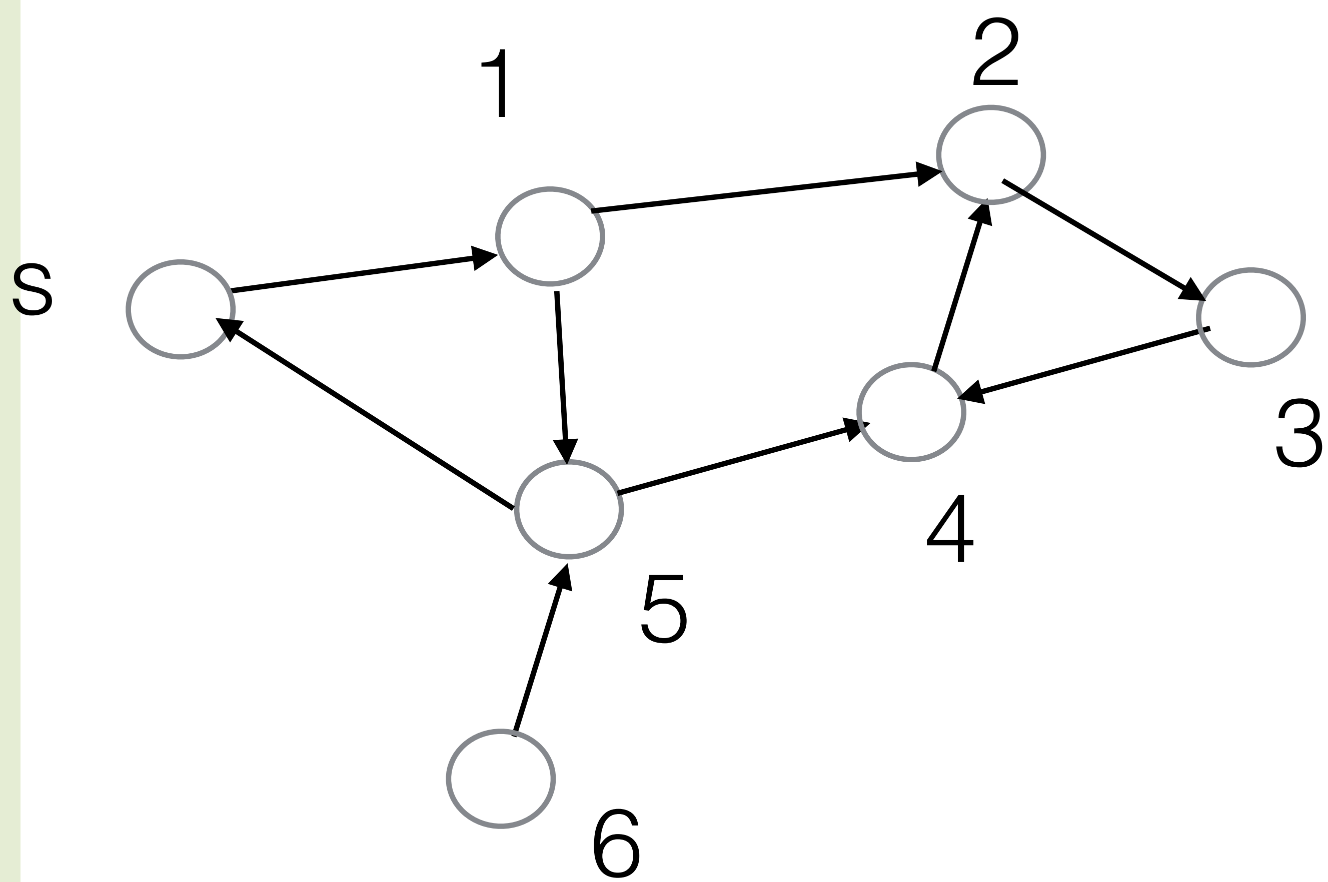
Shortest Paths



- Single source shortest path (one s, all t)



Shortest Paths

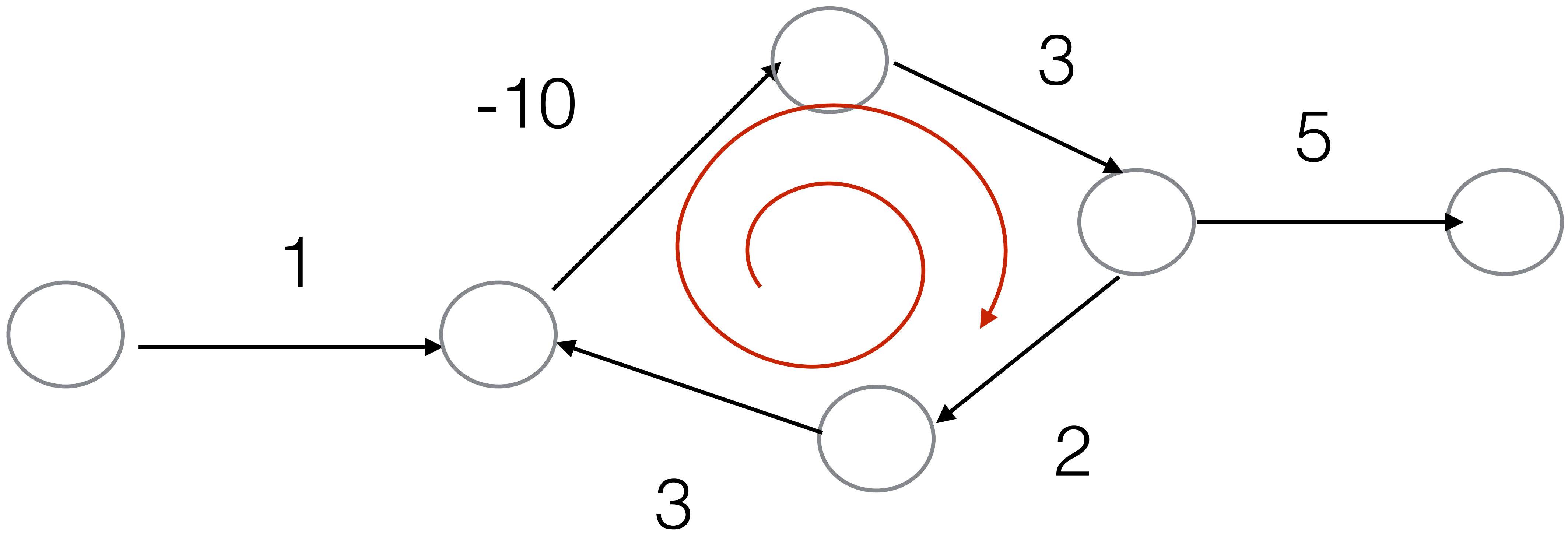


- Single source shortest path (one s, all t)
- All pairs shortest path (all s, all t)

Input = directed graph (V,E) with lengths w(e) on edges

- all w(e) ≥ 0
- some w(e) < 0
- Dijkstra only (?!) works for singe source shortest paths when all weights non-negative (not really…)

Shortest Paths

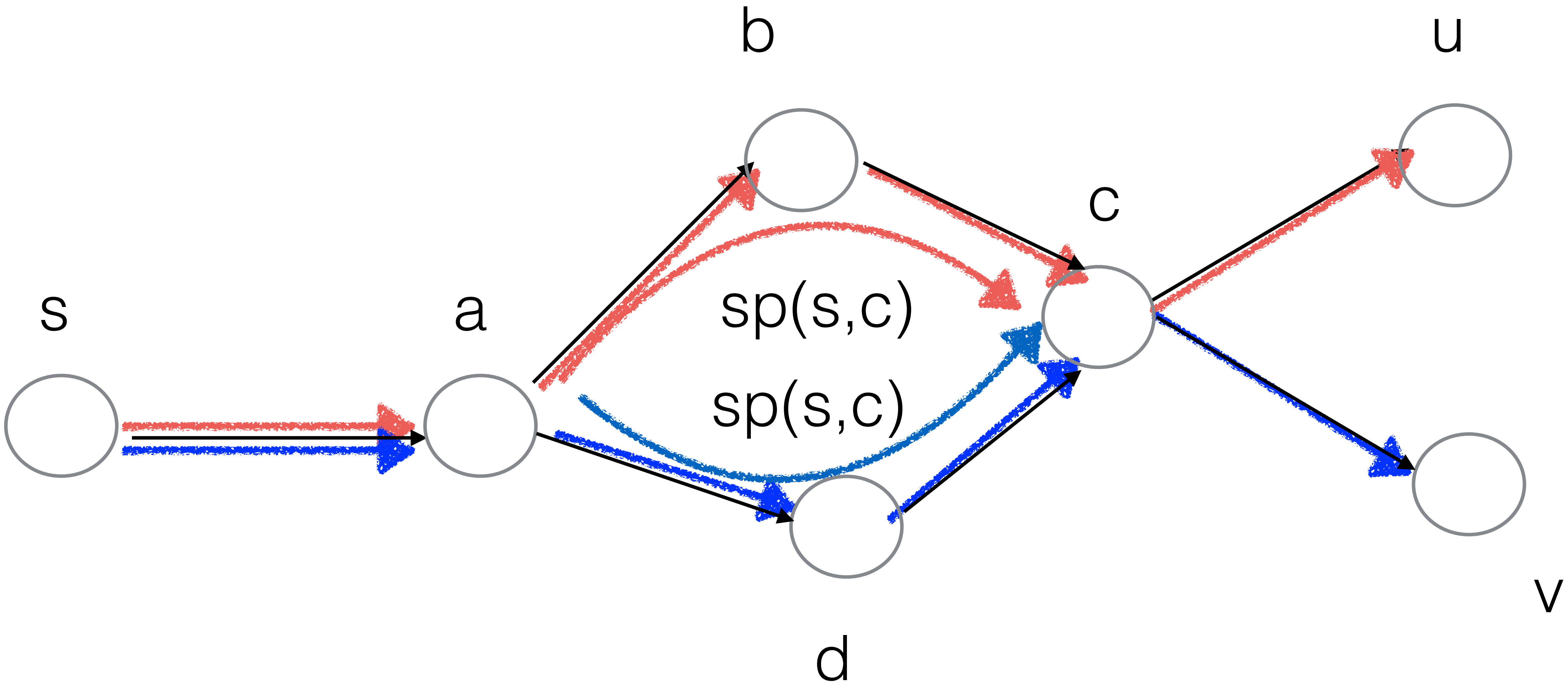


Can we allow arbitrary negative weights?

No shortest path!

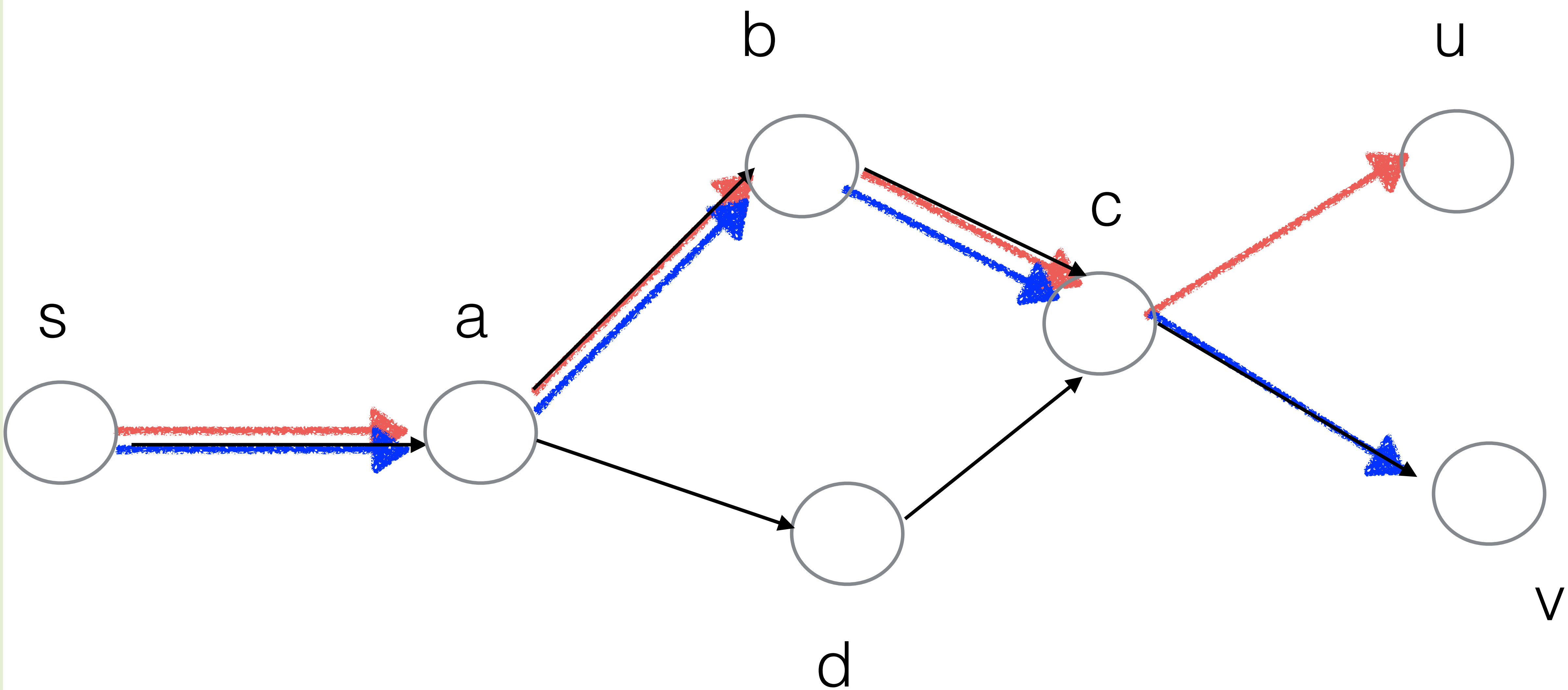
Negative cycles are bad. Assume they don't exist

Shortest Path Trees



If shortest paths are unique they form a tree
what if they are not unique?

Shortest Path Trees



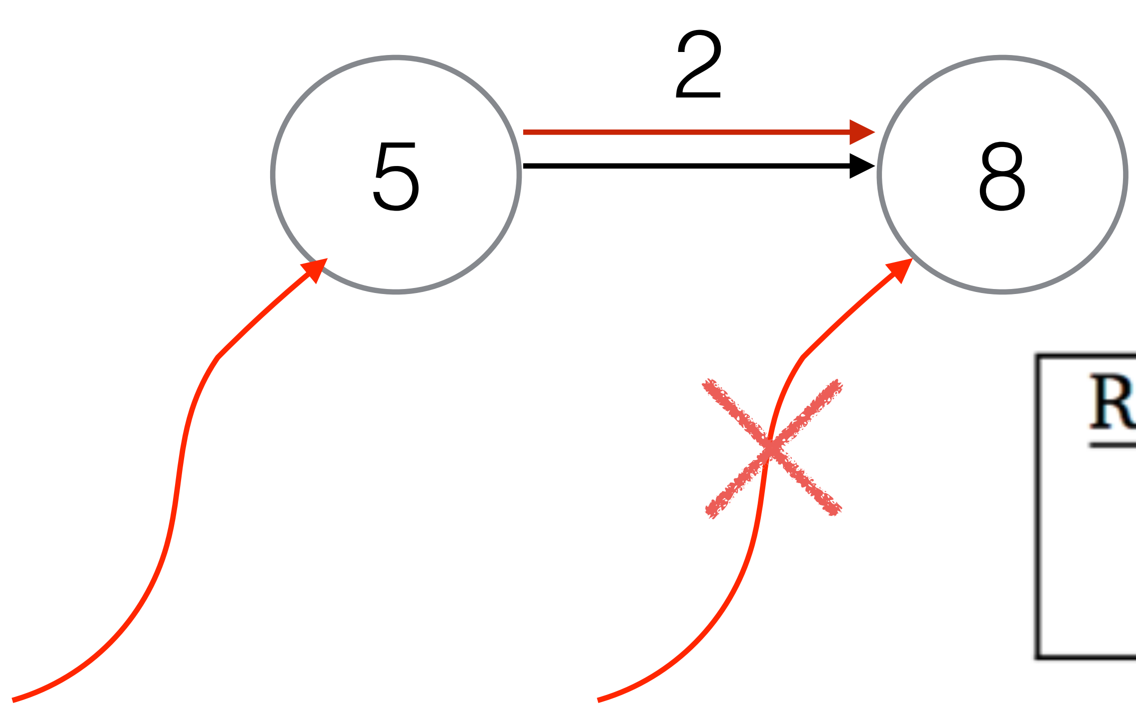
There is a set of shortest paths from s to every vertex that defines a tree

Every SSSP algorithm

Maintain at every vertex:

- dist(v) : the length of the tentative shortest path from s to v or ∞ if there is no such path.
- pred(v): the predecessor of v in the tentative shortest path from s to v or NULL if there is no such vertex.
- think of storing the dist(v) value on the node.

edge $u \to v$ is tense if $dist(v) > dist(u)+w(u \to v)$



<u>RELAX($u \to v$):</u>
$dist(v) \leftarrow dist(u) + w(u \to v)$
$pred(v) \leftarrow u$

Every SSSP algorithm

<u>INITSSSP(s):</u>
    $dist(s) \leftarrow 0$
    $pred(s) \leftarrow$ NULL
    for all vertices $v \neq s$
        $dist(v) \leftarrow \infty$
        $pred(v) \leftarrow$ NULL

If there are no tense edges then for every vertex v, dist(v) is shortest path distance.

While some edges is tense, relax it

edge u → v is tense if
dist(v) > dist(u)+w(u →v)

<u>RELAX($u \rightarrow v$):</u>
    $dist(v) \leftarrow dist(u) + w(u \rightarrow v)$
    $pred(v) \leftarrow u$

Every SSSP algorithm

```
INITSSSP(s):
  dist(s) ← 0
  pred(s) ← NULL
  for all vertices v ≠ s
    dist(v) ← ∞
    pred(v) ← NULL
```

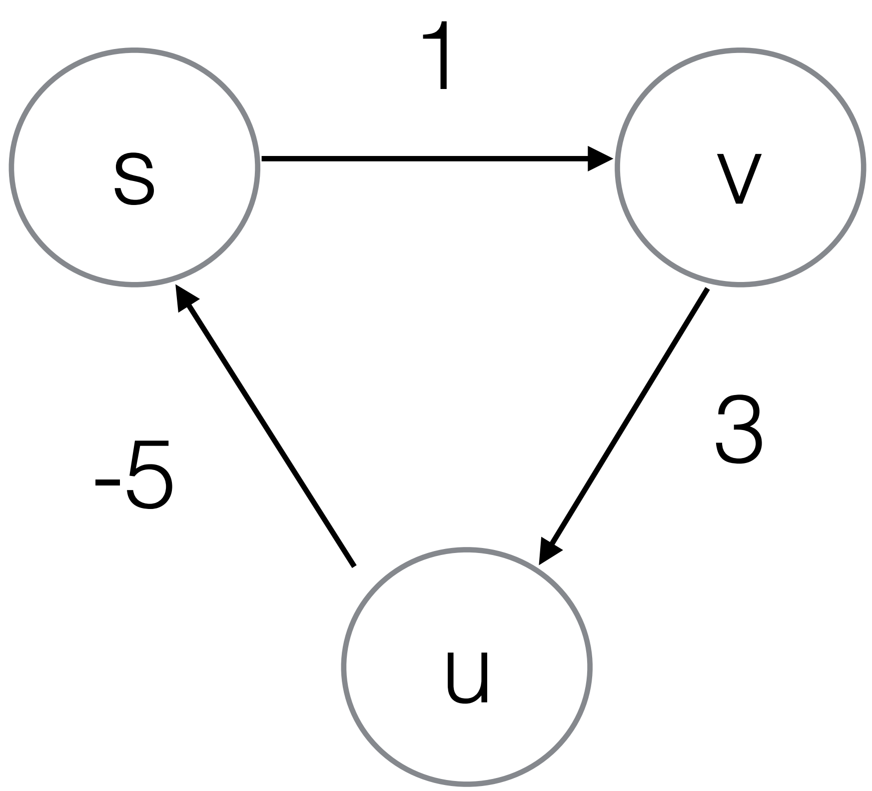
While some edges is tense,
relax it

makes no assumption on negative weights.
Does assume no negative cycle (how?).

Every SSSP algorithm

```
InitSSSP(s):
    dist(s) ← 0
    pred(s) ← Null
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← Null
```

While some edges is tense,
relax it

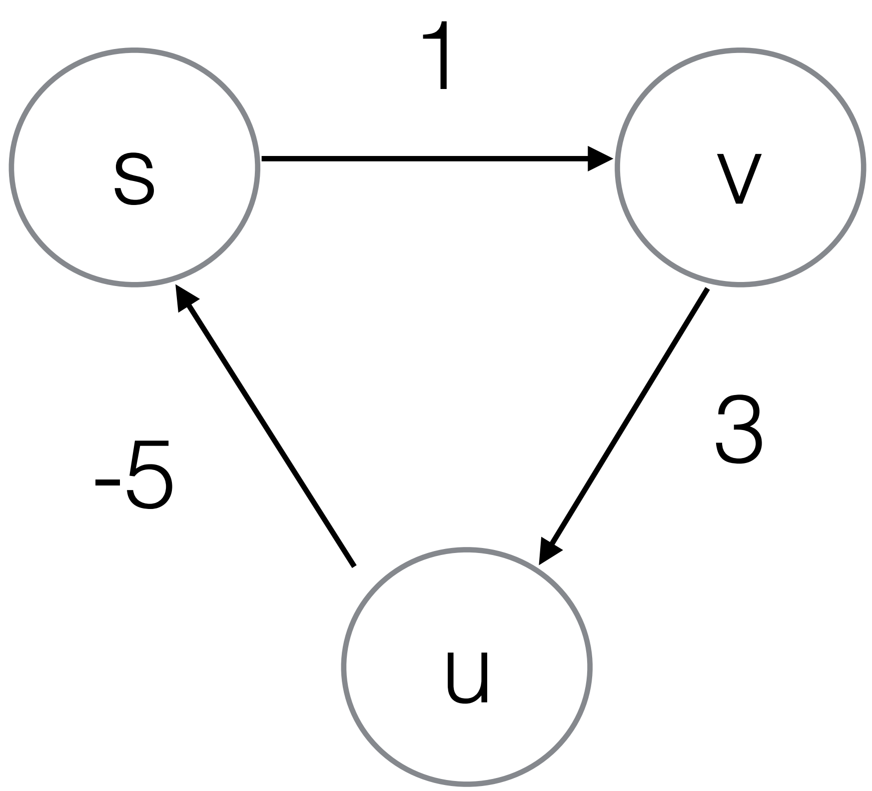


dist(s) = 0
dist(v) = ∞
dist(u) = ∞

Every SSSP algorithm

<u>INITSSSP(s):</u>
  $dist(s) \leftarrow 0$
  $pred(s) \leftarrow$ NULL
  for all vertices $v \neq s$
    $dist(v) \leftarrow \infty$
    $pred(v) \leftarrow$ NULL

While some edges is tense, relax it

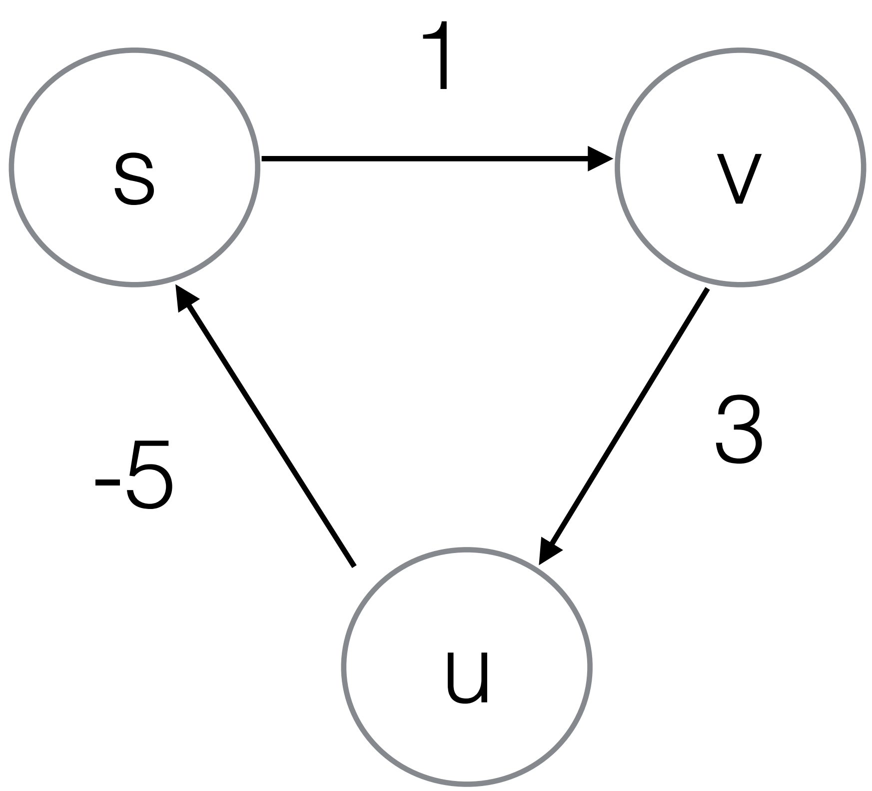


dist(s) = 0
dist(v) = 1
dist(u) = ∞

Every SSSP algorithm

<u>INITSSSP(s):</u>
  $dist(s) \leftarrow 0$
  $pred(s) \leftarrow$ NULL
  for all vertices $v \neq s$
    $dist(v) \leftarrow \infty$
    $pred(v) \leftarrow$ NULL

While some edges is tense,
relax it

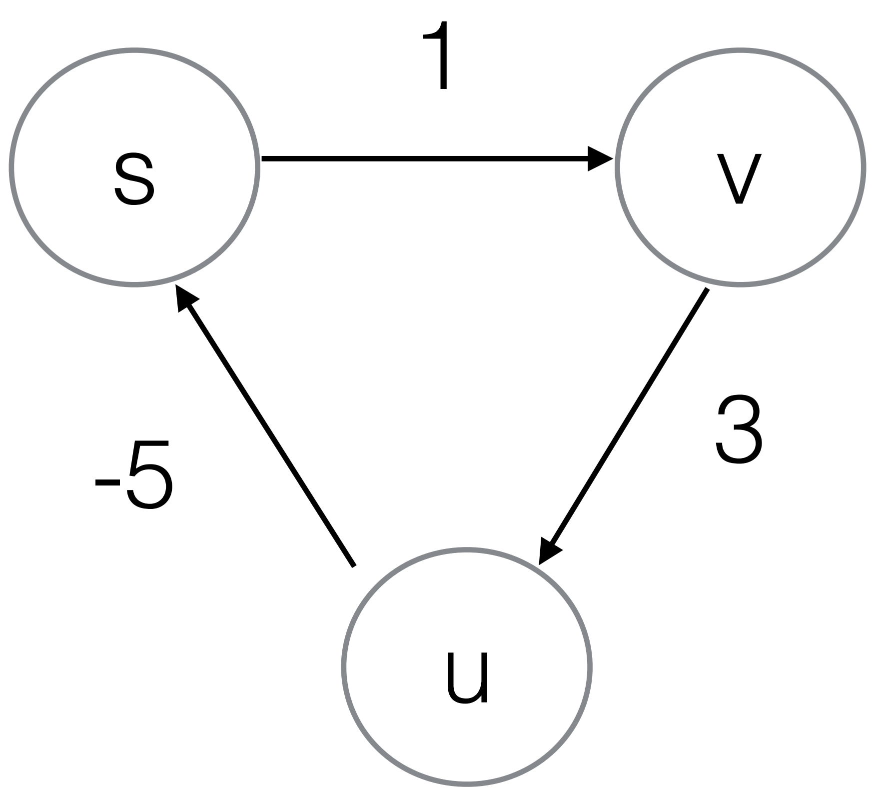


dist(s) = 0
dist(v) = 1
dist(u) = 4

Every SSSP algorithm

<u>INITSSSP(s):</u>
  $dist(s) \leftarrow 0$
  $pred(s) \leftarrow$ NULL
  for all vertices $v \neq s$
    $dist(v) \leftarrow \infty$
    $pred(v) \leftarrow$ NULL

While some edges is tense,
relax it

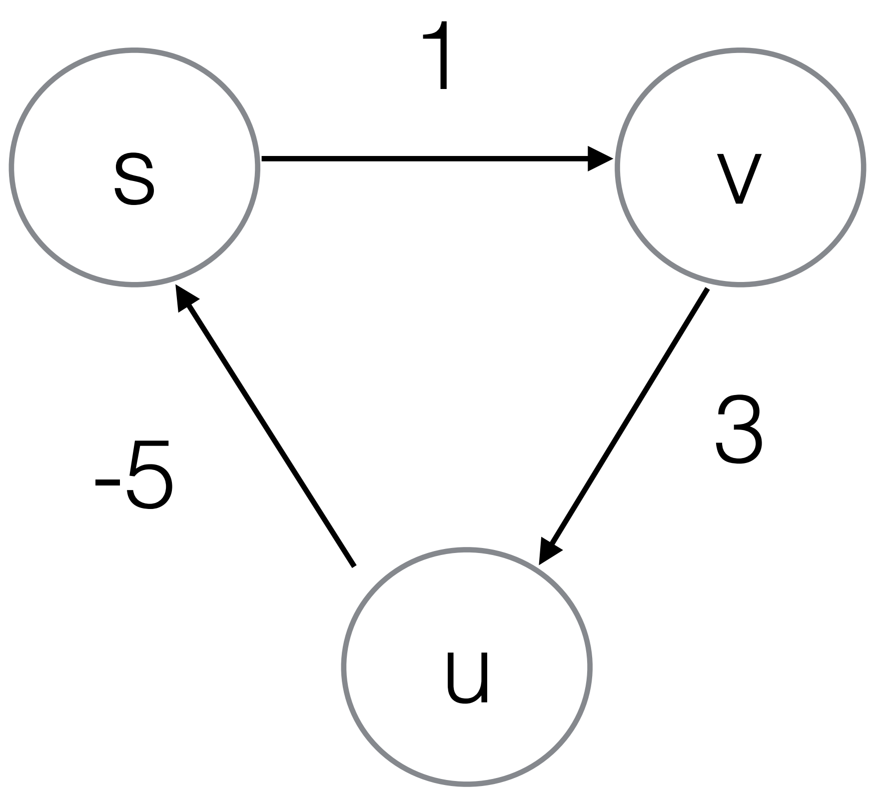


dist(s) = -1
dist(v) = 1
dist(u) = 4

Every SSSP algorithm

<u>InitSSSP(*s*):</u>
$dist(s) \leftarrow 0$
$pred(s) \leftarrow$ Null
for all vertices $v \neq s$
  $dist(v) \leftarrow \infty$
  $pred(v) \leftarrow$ Null

While some edges is tense,
relax it



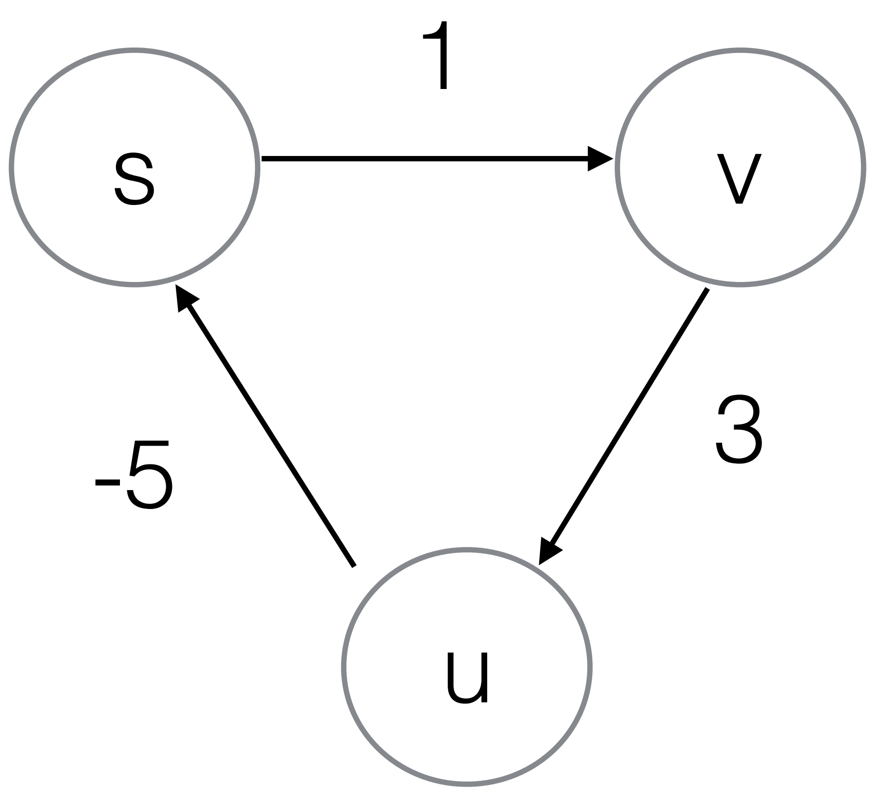
dist(s) = -1
dist(v) = 0
dist(u) = 4

Every SSSP algorithm

```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

Ford ('53)

While some edges is tense, relax it → runs into infinite loop



dist(s) = -1
dist(v) = 0
dist(u) = 3

some edge always tense!

Every SSSP algorithm

```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

```
GENERICSSSP(s):
    INITSSSP(s)
    put s in the bag
    while the bag is not empty
        take u from the bag
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                put v in the bag
```

Without specifying how to find tense edges, not an algorithm

weird thing about it: a vertex might be put into bag multiple times

Every SSSP algorithm

```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

```
GENERICSSSP(s):
    INITSSSP(s)
    put s in the bag
    while the bag is not empty
        take u from the bag
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                put v in the bag
```

What data structure?

queue, stack? (both give correct algo, but maybe exp time)

Every SSSP algorithm

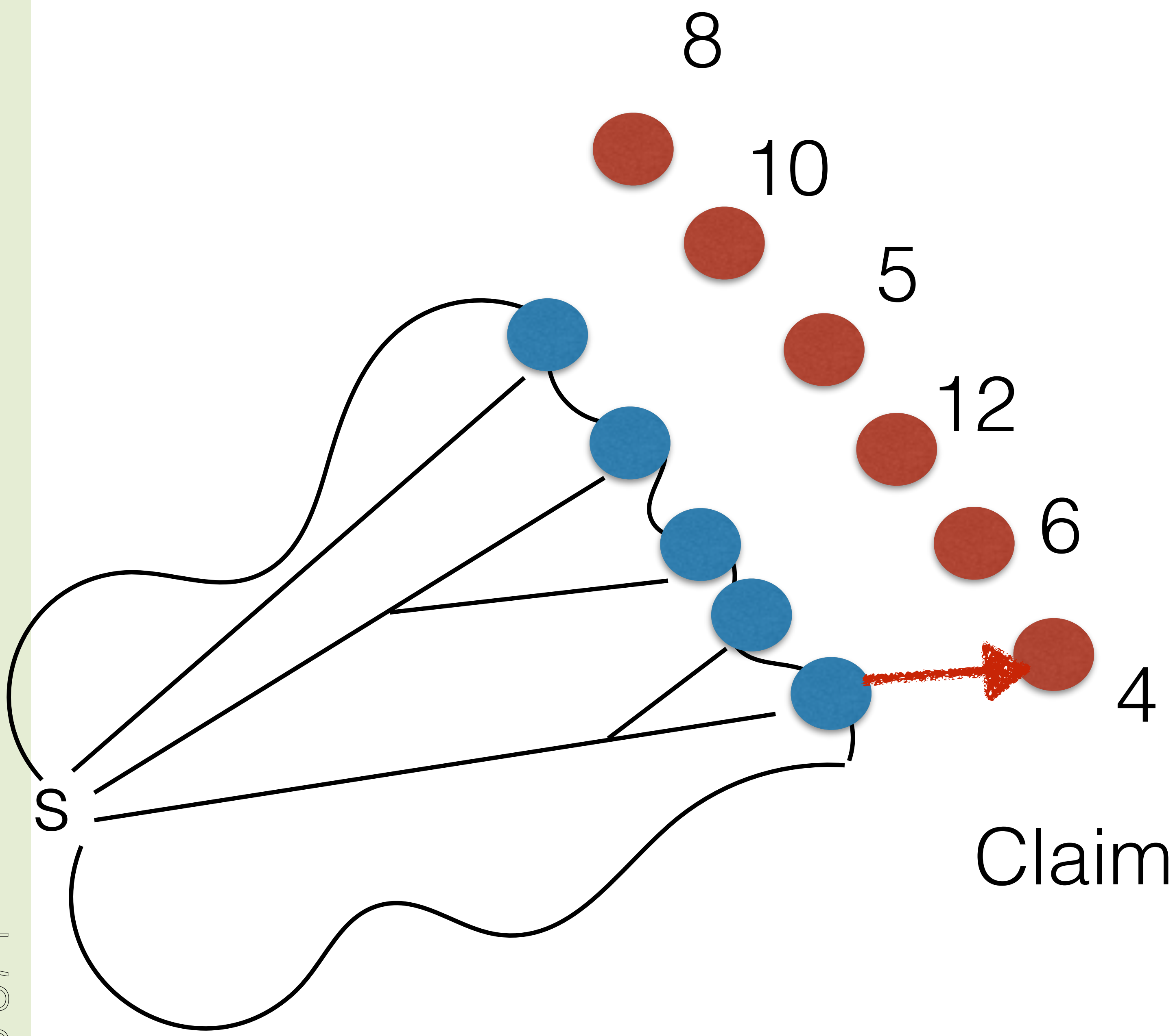
```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

```
GENERICSSSP(s):
    INITSSSP(s)
    put s in the bag
    while the bag is not empty
        take u from the bag
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                put v in the bag
```

Dijkstra: Priority Queue
increasing order of their shortest path distance.
Every vertex is visited exactly once, and when that happens the distance is correct

Dijkstra

assume I have computed a partial shortest path tree



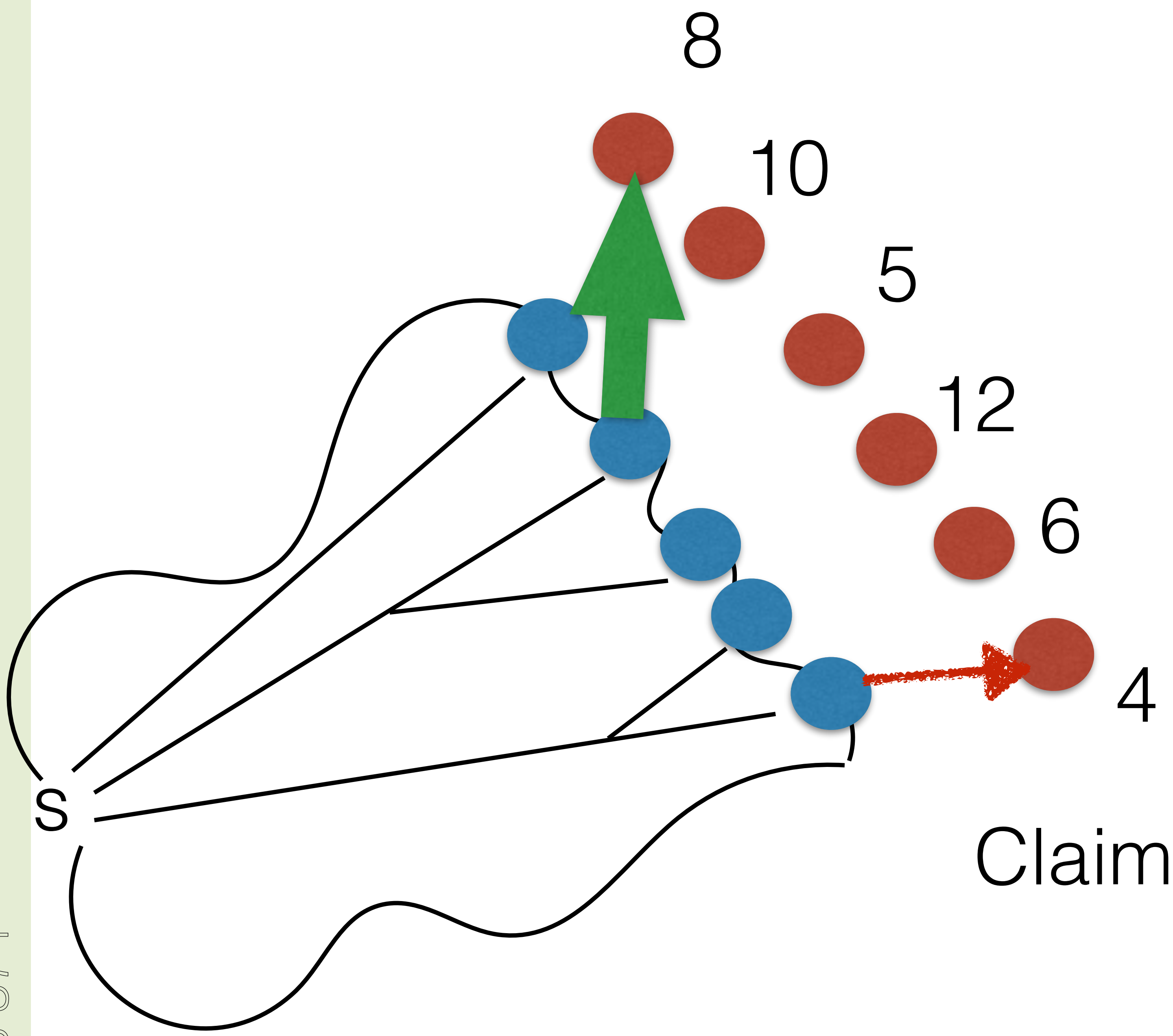
consider the edges from partial tree to all red vertices

what edge to choose in order to extend the tree?

Claim: this edge is in the tree

Dijkstra

assume I have computed a partial shortest path tree



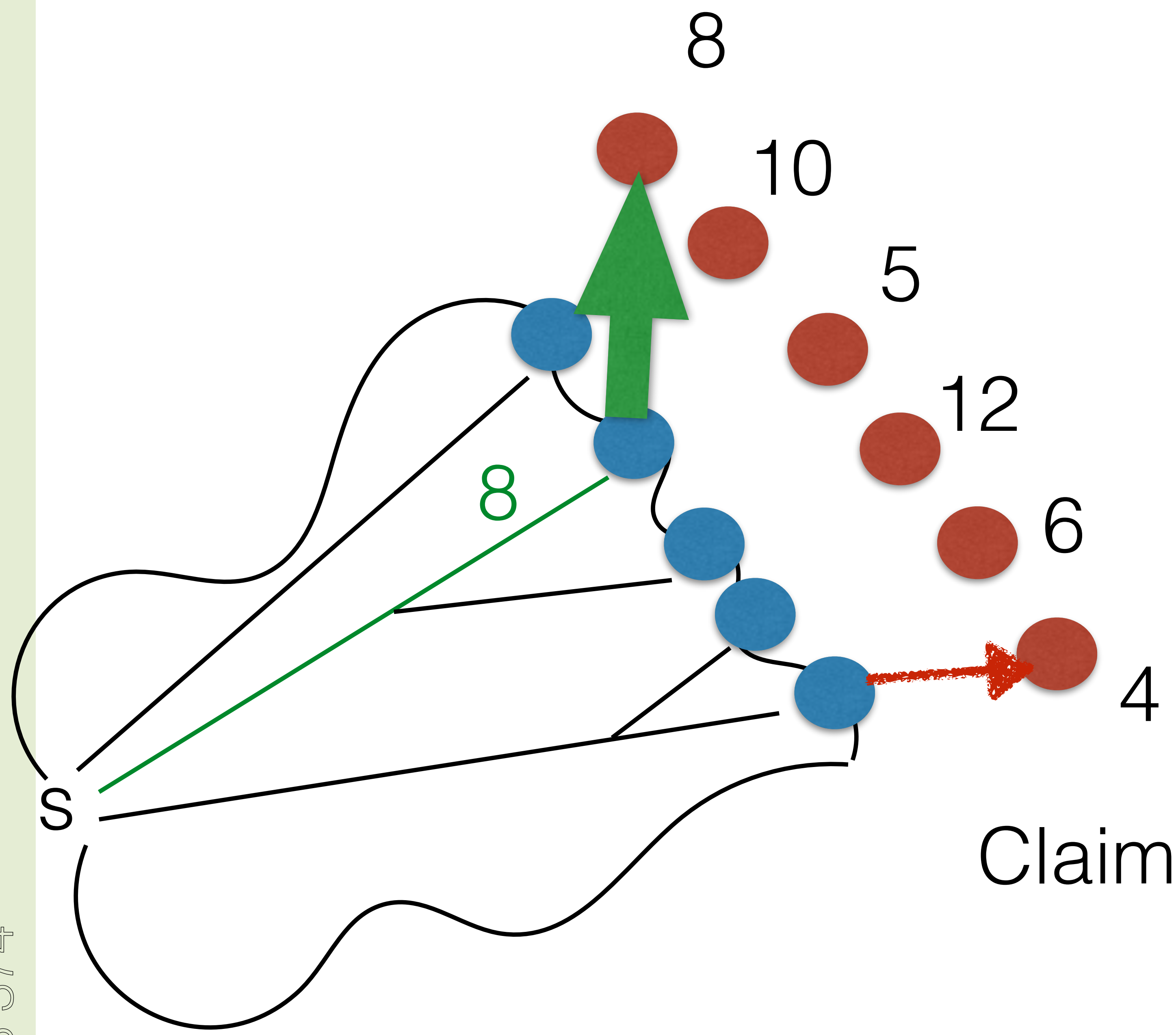
consider the edges from partial tree to all red vertices

what edge to choose in order to extend the tree?

Claim: this edge is in the tree

Dijkstra

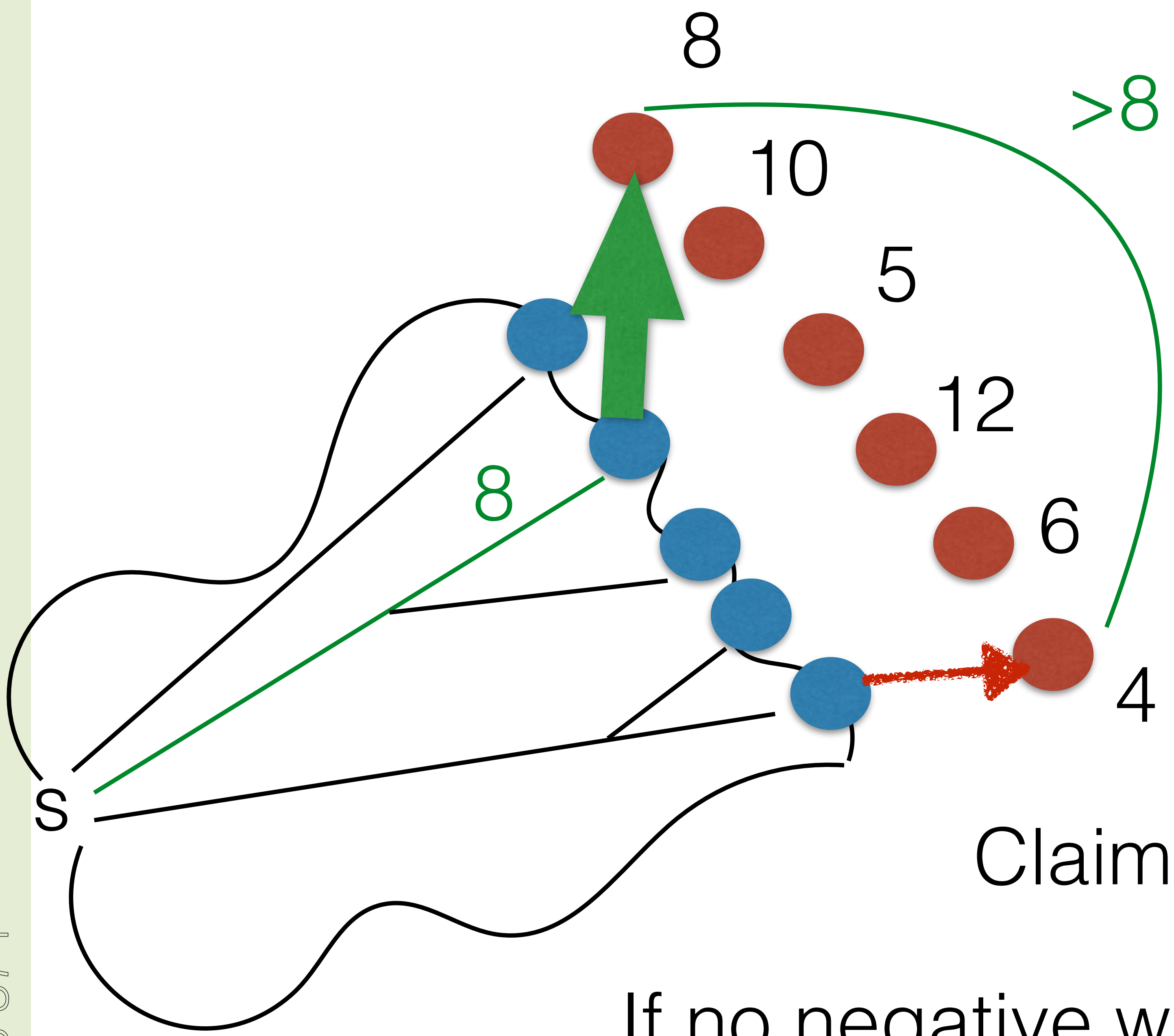
assume I have computed a partial shortest path tree



Claim: this edge is in the tree

Dijkstra

assume I have computed a partial shortest path tree



Claim: this edge is in the tree

If no negative weights, Dijkstra is greedy!

Dijkstra

a.k.a "Closest first search"

Algorithm:

if all $w(e) \geq 0$ then

each node leaves priority queue once

≤ 1 priority queue operation per edge

$O(|E|\log V)$

if there is $w(e) < 0$ then

$O(2^{|V|})$ time

Every SSSP algorithm

```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

```
GENERICSSSP(s):
    INITSSSP(s)
    put s in the bag
    while the bag is not empty
        take u from the bag
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                put v in the bag
```

Difference between Dijkstra and Generic?